GSFC/CP—1998–206860

Second International Workshop on Software Engineering and Code Design in Parallel Meteorological and Oceanographic Applications

Matthew O'Keefe, Christopher Kerr, Editors

Proceedings of a workshop sponsored by the U.S. Department of Energy, Office of Biological and Environmental Research; the Department of Defense, High Performance Computing and Modernization Office; and the NASA Goddard Space Flight Center, Seasonal-to-Interannual Prediction Project, and held at the Camelback Inn, Scottsdale, Arizona June 15–18, 1998

National Aeronautics and
Space Administration

Goddard Space Flight Center
Greenbelt, Maryland 20771

June 1998

The NASA STI Program Office ... in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA Scientific and Technical Information (STI) Program Office plays a key part in helping NASA maintain this important role.

The NASA STI Program Office is operated by Langley Research Center, the lead center for NASA's scientific and technical information. The NASA STI Program Office provides access to the NASA STI Database, the largest collection of aeronautical and space science STI in the world. The Program Office is also NASA's institutional mechanism for disseminating the results of its research and development activities. These results are published by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA's counterpart of peer-reviewed formal professional papers but has less stringent limitations on manuscript length and extent of graphic presentations.
- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.
- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.
- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or cosponsored by NASA.
- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and mission, often concerned with subjects having substantial public interest.
- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services that complement the STI Program Office's diverse offerings include creating custom thesauri, building customized databases, organizing and publishing research results . . . even providing videos.

For more information about the NASA STI Program Office, see the following:

- Access the NASA STI Program Home Page at http://www.sti.nasa.gov/STI-homepage.html
- E-mail your question via the Internet to help@sti.nasa.gov
- Fax your question to the NASA Access Help Desk at (301) 621-0134
- Telephone the NASA Access Help Desk at (301) 621-0390
- Write to:
  NASA Access Help Desk
  NASA Center for AeroSpace Information
  7121 Standard Drive
  Hanover, MD 21076-1320

GSFC/CP—1998–206860

Second International Workshop on Software Engineering and Code Design in Parallel Meteorological and Oceanographic Applications

Matthew O'Keefe, University of Minnesota
Christopher Kerr, International Business Machines

Proceedings of a workshop sponsored by the U.S. Department of Energy, Office of Biological and Environmental Research; the Department of Defense, High Performance Computing and Modernization Office; and the NASA Goddard Space Flight Center, Seasonal-to-Interannual Prediction Project, and held at the Camelback Inn, Scottsdale, Arizona June 15–18, 1998

National Aeronautics and
Space Administration

Goddard Space Flight Center
Greenbelt, Maryland 20771

June 1998

Available from:

NASA Center for AeroSpace Information
800 Elkridge Landing Road
Linthicum Heights, MD 21090-2934
Price Code: A17

National Technical Information Service
5285 Port Royal Road
Springfield, VA 22161
Price Code: A10

Workshop Organizing Committee

Matthew O'Keefe
University of Minnesota

Christopher Kerr
International Business Machines

Workshop Coordinator

Wendy Marshall
Geophysical Fluid Dynamics Laboratory/NOAA

Scientific Program Committee

David Dent
European Center for Medium Range Weather Forecasts

Steven Hammond
National Center for Atmospheric Research

Matthew O'Keefe
University of Minnesota

Christopher Kerr
International Business Machines

Luke Lonergan
High Performance Technologies Inc.

Robert Malone
Los Alamos National Laboratory

John Michalakes
Argonne National Laboratory

Max Suarez
NASA/Goddard Space Flight Center

Steven Thomas
Environnement Canada

Alan Wallcraft
NRL/Stennis Space Center

Workshop Sponsors

U.S. DEPARTMENT OF ENERGY,
OFFICE OF BIOLOGICAL AND ENVIRONMENTAL RESEARCH

U.S. DEPARTMENT OF DEFENSE,
HIGH PERFORMANCE COMPUTING AND MODERNIZATION OFFICE

U.S. NATIONAL AERONAUTICS AND SPACE ADMINISTRATION,
SEASONAL-TO-INTERANNUAL PREDICTION PROJECT, GODDARD SPACE FLIGHT CENTER

Workshop Schedule

Monday Morning, June 15, 1998

8:00 am		Registration and Breakfast
8:45 am	Matthew O'Keefe, University of Minnesota Christopher Kerr, IBM	Welcome and Opening Remarks
Session 1	**Atmospheric Models** Session Chairperson: Steven Thomas	
9:00 am	Mike Desgagne, Environnement Canada	Performance of MC2 and the ECMWF IFS Forecast Model on the Fujitsu VPP700 and NEC SX-4M 1
9:45 am		Break with Refreshments
10:00 am	George Mozdzynski, ECMWF	Parallelization of the ECMWF Integrated Forecasting System 13
10:45 am	Bertrand Denis, Canadian Centre for Climate Modelling and Analysis	Coarse-grain Parallelization of the CCCma Atmospheric GCM on a NEC SX-4 15
11:30 am	Richard S. Hemler, GFDL	Key Elements of the User-Friendly, GFDL SKYHI General Circulation Model 29
12:15 pm		Lunch

Monday Afternoon, June 15, 1998

Session 2	**Atmospheric Models** Session Chairperson: Steven Hammond	
1:30 pm	Daniel S. Schaffer, NASA/Goddard Space Flight Center	Design and Performance Analysis of a Massively Parallel Atmospheric General Circulation Model 45
2:15 pm	Ulrich Schaettler, Deutscher Wetterdienst	Requirements and Problems in Parallel Model Development at DWD 57
3:00 pm	Frederick Rawlins, UK Meteorological Office	Experiences with Parallelisation of the Unified Model at the UK Meteorological Office 69
3:45 pm		Break with Refreshments
4:00 pm	Steve Thomas, Environnement Canada	Optimizing MC2 for RISC Architectures: Forecast Accuracy versus Performance 77

Tuesday Morning, June 16, 1998

Tuesday Afternoon, June 16, 1998

Wednesday Morning, June 17, 1998

Wednesday Afternoon, June 17, 1998

Thursday Morning, June 18, 1998

Thursday Afternoon, June 18, 1998

PERFORMANCE OF MC2 AND THE ECMWF IFS FORECAST MODEL ON THE FUJITSU VPP700 AND NEC SX-4M

M. Desgagné, D. Dent, S. Thomas, M. Valin, G. Mozdzynski

Recherche en prévision numérique (RPN), Environment Canada,
2121, route Transcanadienne, Dorval, Québec H9P 1J3, CANADA
michel.desgagne@ec.gc.ca, steve.thomas@ec.gc.ca
michel.valin@ec.gc.ca
Tel. 1-514-421-4661, FAX 1-514-421-2106

European Centre for Medium Range Weather Forecasts
Shinfield Park, Reading, ENGLAND RG2 9AX
david.dent@ecmwf.int, george.mozdzynski@ecmwf.int

Abstract

The NEC SX-4M cluster and Fujitsu VPP700 supercomputers are both based on custom vector processors using low-power CMOS technology. Their basic architectures and programming models are however somewhat different. A multi-node SX-4M cluster contains up to 32 processors per shared memory node, with a maximum of 16 nodes connected via the proprietary NEC IXS fibre channel crossbar network. A hybrid combination of inter-node MPI message-passing with intra-node multitasking or threads is possible. The Fujitsu VPP700 is a fully distributed-memory vector machine with a scalable crossbar interconnect which also supports MPI. The parallel performance of the MC2 model for high-resolution mesoscale forecasting over large domains and of the IFS RAPS 4.0 benchmark are presented for several different machine configurations. These include an SX-4/32 with 8 GB main memory unit (MMU), an SX-4/32M cluster (SX-4/16, 8 GB MMU + SX-4/16, 4 GB MMU) and up to 80 PE's of the VPP700.

1. Introduction

John Hennessy, professor of computer science, dean of the Stanford University School of Engineering and co-inventor of the MIPS RISC microprocessor recently speculated during the Supercomputing 97 conference in San Jose that vector processors would disappear from high-performance computing within five to ten years [4]. Given the impressive sustained floating point execution rates of the NEC SX-4 and Fujitsu VPP700 vector processors, these two Japanese computer vendors could easily argue that 'reports of their demise are greatly exaggerated'. Despite the fact that the peak execution rates of pipelined RISC microprocessors continue to double every eighteen months, highly optimized codes can usually sustain no more than 15 to 20% of peak. This situation may change as larger secondary cache memories become available. However, the SX-4 vector processor can routinely achieve 1 Gflops/sec or higher on representative atmosphere, ocean and climate codes. Indeed, both SX-4 and VPP700 processors can sustain in the range of 30 to 50% of their rated peak performance levels. Both NEC and Fujitsu build scalable parallel architectures based on these processors with existing or planned customer installations capable of 100 Gflops/sec or higher sustained performance.

Cluster type architectures are becoming prevalent in high-performance computing and current designs can trace their roots back to the pioneering work of Paul Woodward who demonstrated the capabilities of symmetric multiprocessor (SMP) cluster supercomputing in 1993 [9]. The US Department of Energy's Accelerated Strategic Computing Initiative (ASCI) has also led to the announcement of cluster type computers from several US manufacturers. Individual nodes contain from 1 to 128 RISC/cache or vector processors. Typically, shared or distributed-shared memory (DSM) is used within a node and additional cache-coherence mechanisms are often present. Low-latency, high-bandwidth interconnection networks then link these nodes together. NEC SX-4M clusters and the Fujitsu VPP700 perhaps represent opposite ends of the design spectrum. SX-4 nodes contain up to 32 vector processors and 8 Gbytes of fast SSRAM main memory, whereas the VPP700 is a fully distributed-memory machine. Each VPP700 processing element contains a vector processor along with up to 2 Gbytes of slower SDRAM memory. The two machines are compared in this paper by using benchmarks of two decidely different atmospheric models. The ECMWF IFS forecast model is a global weather prediction model based on the spectral transform method. The Canadian MC2 is a nonhydrostatic, fully compressible limited area atmospheric model designed for high-resolution mesoscale forecasting. A fully 3D semi-implicit scheme is implemented with second-order finite differences in space. Both models implement semi-Lagrangian advection with overlaps.

2. The NEC SX-4M and Fujitsu VPP700

The multi-node NEC SX-4M is an SMP cluster type architecture with up to 32 processors per node and a maximum of 16 nodes interconnected via the proprietary NEC IXS crossbar network with fibre channel interface. Each node executes an enhanced version of UNIX System V with features such as resource sharing groups (RSG) to dedicate resources to single or multi-node jobs. The total 8 Gbytes/sec IXS (bi-directional) bandwidth is augmented by a direct memory-mapped addressing scheme between nodes [3]. Each CPU of the SX-4 contains a scalar unit and a vector unit. The vector processor is based on low-power CMOS technology with a clock cycle time of 8ns (125MHz). Three floating point formats are supported: IEEE 754, Cray, and IBM. The vector unit of each processor consists of 8 parallel sets of 4 vector pipelines, 1 add/shift, 1 multiply, 1 divide, and 1 logical. For each vector unit there are 8 64-bit vector arithmetic registers and 64 64-bit vector data registers used as temporary space. The peak performance of a concurrent vector add and vector multiply is 2 Gflops/sec and atmospheric codes can sustain 1 Gflops/sec or higher. Main Memory Unit (MMU) configurations for a node range from 512 Mbytes to 8 Gbytes of 15ns Synchronous Static Random Access Memory (SSRAM). The maximum 8 Gbytes configuration comprises 32 banks of 256 Mbytes each, providing memory bandwidths of 16 Gbytes/sec per processor. Supplementing main memory is 16 or 32 Gbytes of eXtended Memory Unit (XMU) built with 60ns Dynamic Random Access Memory (DRAM) and having a 4 Gbyte/sec bandwidth. MPI/SX is based on a port of the MPICH package by NEC's C & C European Lab with the assistance of Rusty Lusk and Bill Gropp from Argonne National Laboratory [3].

A processing element of the Fujitsu VPP700 also contains both a scalar and vector unit. The vector unit consists of 8 functional units which can operate in parallel. The peak performance of the vector unit is 2.2 Gflops/sec, whereas the scalar unit is a 100 Mflops/sec processor. Both 32 and 64-bit IEEE floating point formats are supported. Each PE can be configured with up to 2 Gbytes of Synchronous Dynamic Random Access Memory (SDRAM). A full copy of the 32-bit UNIX operating system kernel is executed by each processor with 1.7 Gbytes available for programs and data. A 64-bit operating system is planned for the next generation VPP architecture with up to 8 Gbytes of memory per PE. Processing elements are interconnected with a scalable crossbar switching network, capable of 570 Mbytes/sec (bi-directional) point-to-point transfer rates. MPI is implemented on top of the proprietary VPP message-passing layer. Any processor can make I/O requests but only 11 of the 116 VPP700 PE's at the ECMWF (the so-called I/O processors) are configured with disks.

3. Parallel Programming Models

Climate and ocean modeling groups at NCAR [5] and the University of Minnesota [6] have identified and tested hybrid programming models for SMP architectures. Shared-memory tasking mechanisms or threads can be applied for intra-node parallelism, whereas inter-node communication is implemented with MPI. Coarse-grain tasks on an SX-4 node are created with the **pt_fork** and **pt_join** primitives and loop-level parallelism in the form of micro-tasking is specified through the inline compiler directive *vdir pardo*. A POSIX threads compliant library **pt_thread** is also available. With the recent acceptance of an OpenMP standard for shared-memory parallelism, it should now be possible to build portable codes employing both MPI and auto-tasking. The MC2 model is discretised on a $N_X \times N_Y \times N_Z$ grid, where the number of points in the vertical direction is typically one order of magnitude less than in the horizontal. A distributed-memory model of computation is based on a domain decomposition across a $P_X \times P_Y$ processor mesh. All vertical loops in the dynamics and physics code are micro-tasked, allowing for a hybrid combination with boundary exchanges implemented using MPI. The elliptic solver in MC2 is a minimal residual Krylov iteration with line relaxation preconditioners (see Skamarock et al. [7] and Thomas et al. [8]). To handle global data dependencies, a data transposition strategy is implemented using MPI all-to-all communication. Fixed-size halos are implemented for semi-Lagrangian advection.

The IFS forecast model is a global spectral model which can use either a full or reduced Gaussian grid. In the case of a reduced grid, the number of grid points along a latitude line decreases near the poles. Both Eulerian and semi-Lagrangian advection schemes are available. A parallel domain decomposition is based on a latitude by longitude decomposition in grid point, Fourier and spectral space where NPROC = NPROCA $\times$ NPROCB. A data transposition strategy is implemented between each computational phase of a time-step. A fixed overlap strategy is also implemented for the distributed-memory implementation of semi-Lagrangian advection where the global maximum wind-speed determines the halo size (see Dent and Mozdzynski [2]). The shared-memory version of the model is still retained and was not sacrificed in order to build a distributed-memory implementation. In fact, the IFS model can be run in a hybrid shared/distributed configuration. FFT's are computed on all processors and are independent in both the vertical and longitudinal directions. Likewise, the Legendre transforms are also executed on all processors and are independent in the vertical and over spectral waves. Finally, the IFS has been coded to perform effectively on both vector and RISC/cache architectures by supporting a runtime parameter NPROMA which controls locality of reference.

4. Benchmark Results

We have benchmarked the full forecast configurations of the MC2 (adiabatic kernel + RPN physics version 3.5) and IFS models at the CMC in Montreal and at the ECMWF in Reading. The current CMC configuration consists of the operational machine 'hiru', an SX-4/32 with 8 Gbyte MMU along with 'yonaka' (SX-4/16 + 4 GB MMU) and 'asa' (SX-4/16 + 8 GB MMU). The two SX-4/16 nodes can operate as an SX-4/32M cluster and all three machines will be connected to the IXS crossbar in 1998. Four full nodes in an SX-4/128M cluster should be in place by the year 2000 or 2001, with a peak performance of 256 Gflops/sec. Given our results to date, it is reasonable to expect that 50% of peak is possible on such a machine. The ECMWF VPP700 is currently configured with 116 PE's, each containing 2 Gbytes of memory or 232 Gbytes in total.

The MC2 model is written in Fortran 77 with Cray `POINTER` extensions for dynamic memory allocation. The code was compiled using 32-bit arithmetic on both the SX-4 and VPP700. Whereas the IBM floating point format was specified on the SX-4, 32-bit IEEE arithmetic was used on the VPP700. The only compiler options specified to assist in vectorisation were `-pvctl noassume loopcnt=1000000`. Extensive inline compiler directives such as *vdir nodep* are specified in the physics library due to dynamic memory allocation. The SX-4 compiler is conservative and assumes both aliasing and recurrences are present unless otherwise indicated. The vectorisation level on the SX-4 (scalar versus vector instructions) then usually exceeds 98%. Similar directives were specified to the VPP700 Fortran 90 compiler `frt`. Multi-node SX-4 runs require a `mpi.hosts` file containing the number of processes to launch on each node. In particular, the order of processes launched from this file determines their rank in `MPI_COMM_WORLD`.

The IFS forecast model code is written in a subset of Fortran 90 with extensive use of `ALLOCATABLE` arrays. The model code was compiled for 64-bit IEEE arithmetic on both the SX-4 and VPP700 machines. In fact, this was our first experience at RPN/CMC with the NEC Fortran 90 compiler. It was found to be far too slow for production usage and would likely perform better as a cross-compiler similar to Fujitsu's `frtpx` run on a SGI/Cray Origin 2000 at the ECMWF. Vectorisation and performance of the IFS code are largely determined by the `NAMELIST` parameters `NRPROMA` for the radiation package and `NPROMA` in the dynamics. In all tests we varied `NPROCA` and set `NPROCB=1` [1]. Performance data for the IFS RAPS 4.0 benchmark (T106L19, T213L31) and an MC2 run at 10km resolution using a $512 \times 432 \times 41$ grid ($10 \times \Delta t = 180$sec) are presented at the end of the paper. In both cases, multi-node performance is higher from 8 up to 32 PE's on the SX-4.

5. Discussion and Conclusions

For both the MC2 and IFS models, we encountered what might be best characterized as a problem with 'memory starved' nodes. The SX-4 has 128 Mbytes of SSRAM memory per Gflop of computing power, whereas the VPP700 has over 900 Mbytes of SDRAM per Gflop, a factor of 7 more in terms of memory size. In the case of the SX-4, it appears that 8 Gbytes of fast SSRAM may not be sufficient for 32 processors, each operating at 1 Gflop/sec, in a single distributed-memory program. Since the SX-4 is a 'transition' machine, designed to support both a traditional computing mix of single threaded jobs and scalable multi-node applications, certain design compromises were required. Future designs such as the follow-on SX-5 from NEC, or for that matter any SMP cluster type architecture, must strike the right balance between the number of processors per node and providing a memory hierarchy that supports the highest possible sustained execution rate within a node. Shared-memory tasking mechanisms tend to quickly saturate within a node unless very large grain tasks are used. For small problem sizes, a hybrid mix of sub-domain boundary exchanges using MPI combined with micro-tasking in the vertical direction can be efficient. However, we have always found that a distributed-memory model of computation for both inter and intra-node parallelism yields the highest performance and the transition from single to multi-node is seamless across the NEC IXS crossbar switch with no degradation in performance. Moreover, the performance across nodes was better than on a single node and this may be due to memory contention.

Since the scalar units on both the SX-4 and VPP 700 are 20 to 100 times slower than the vector units, scalar code is to be avoided at all costs. With 2 Gbytes of SDRAM available per PE and likely 8 Gbytes in the next generation machine, memory on the VPP 700 is not a major issue. The slower SDRAM may affect the sustainable floating-point execution rate of some scientific codes. Both the SX-4 and VPP700 processors have an abundance of vector registers which the compiler can exploit to reduce memory traffic. We have found in our benchmarks that the SX-4 processor performs slightly better on short vector lengths than the VPP700. The performance of the VPP700 crossbar interconnect for the IFS spectral model is now well documented, but also the particular communication patterns of a grid point model (such as halo exchanges) are also well handled. The overall performance of the IFS forecast model is slightly better on the VPP700 than the SX-4 (both single and multi-node) for the T213L31 benchmark as can be seen from Figure 1. However, the performance is very close and we believe that the gap could be bridged with a modest tuning effort.

References

[1] S. Barros, D. Dent, L. Isaksen, G. Robinson, G. Mozdzynski and F. Wollenweber. The IFS Model: A parallel production weather code. *Parallel Computing*, **21** (1995), pp. 1621-1638.

[2] D. Dent and G. Mozdzynski. ECMWF operational forecasting on a distributed memory platform: Forecast model. Proceedings of the Seventh ECMWF Workshop on the Use of Parallel Processors in Meteorology. eds. G-R Hoffmann and N. Kreitz. World Scientific, Singapore, 1997, pp. 36-51.

[3] R. Hempel, H. Ritzdorf, F. Zimmermann. Implementation of MPI on NEC's SX-4 multi-node architecture. Proceedings of the 4th European PVM–MPI User's Group Meeting. 1997.

[4] J. L. Hennessy. Perspectives on the architecture of scalable multiprocessors: Recent developments and prospects for the future. Presentation, Supercomputing 97, San Jose, November 1997.

[5] R. D. Loft. A modular 3-D dynamical core testbed. Proceedings of the Seventh ECMWF Workshop on the Use of Parallel Processors in Meteorology. eds. G-R Hoffmann and N. Kreitz. World Scientific, Singapore, 1997, pp. 270-283.

[6] A. C. Sawdey and M. T. O'Keefe. A general programming model for developing scalable ocean circulation applications. Proceedings of the Seventh ECMWF Workshop on the Use of Parallel Processors in Meteorology. eds. G-R Hoffmann and N. Kreitz. World Scientific, Singapore, 1997, pp. 209-225.

[7] W. C. Skamarock, P. K. Smolarkiewicz and J. B. Klemp. Preconditioned conjugate-residual solvers for Helmholtz equations in nonhydrostatic models. *Mon. Wea. Rev.*, **125** (1997), pp. 587-599.

[8] S. J. Thomas, A. V. Malevsky, M. Desgagné, R. Benoit, P. Pellerin and M. Valin. Massively parallel implementation of the mesoscale compressible community model. *Parallel Computing*, **23** (1997), 2143-2160.

[9] P. R. Woodward. Perspectives on supercomputing: Three decades of change. *IEEE Computer*, **29** (1996), pp. 99–111. (special 50th anniversary issue).

sl	PEs	real	user	cp	vector	% vec	par
y	1+1	80	78	155	99	97.5	73
y	2+2	71	52	203	129	97.0	65
y	4+4	42	29	227	140	96.5	36
y	12+4	28	16	255	150	96.0	18

Table 1: IFS T106L19 12 hr forecast timings (secs) on SX-4/32M cluster. SX-4/16, 4 GB MMU (yonaka) + SX-4/16, 8 GB MMU (asa). Semi-Lagrangian (y/n). Processing Elements (PEs). Elapsed (real) time. CPU time (user). Total CPU time (cp). Total vector time (vector). Vectorization ratio (% vec). Estimated parallel (par) time.

sl	PEs	real	user	cp	vector	% vec	par
y	4	200	144	566	326	96.0	183
y	8	101	74	580	331	96.0	88
y	16	63	42	665	373	96.0	47
n	2	276	236	465	161	90.0	249
n	4	137	107	420	166	90.0	120
n	6+2	78	55	437	183	90.0	63
n	8	80	57	450	192	90.0	67
n	16	52	33	514	223	89.0	35

Table 2: IFS T213L31 12 hr forecast timings (secs) on SX-4/32, 8 GB MMU (hiru) and SX-4/32M cluster. SX-4/16, 4 GB MMU (yonaka) + SX-4/16, 8 GB MMU (asa).

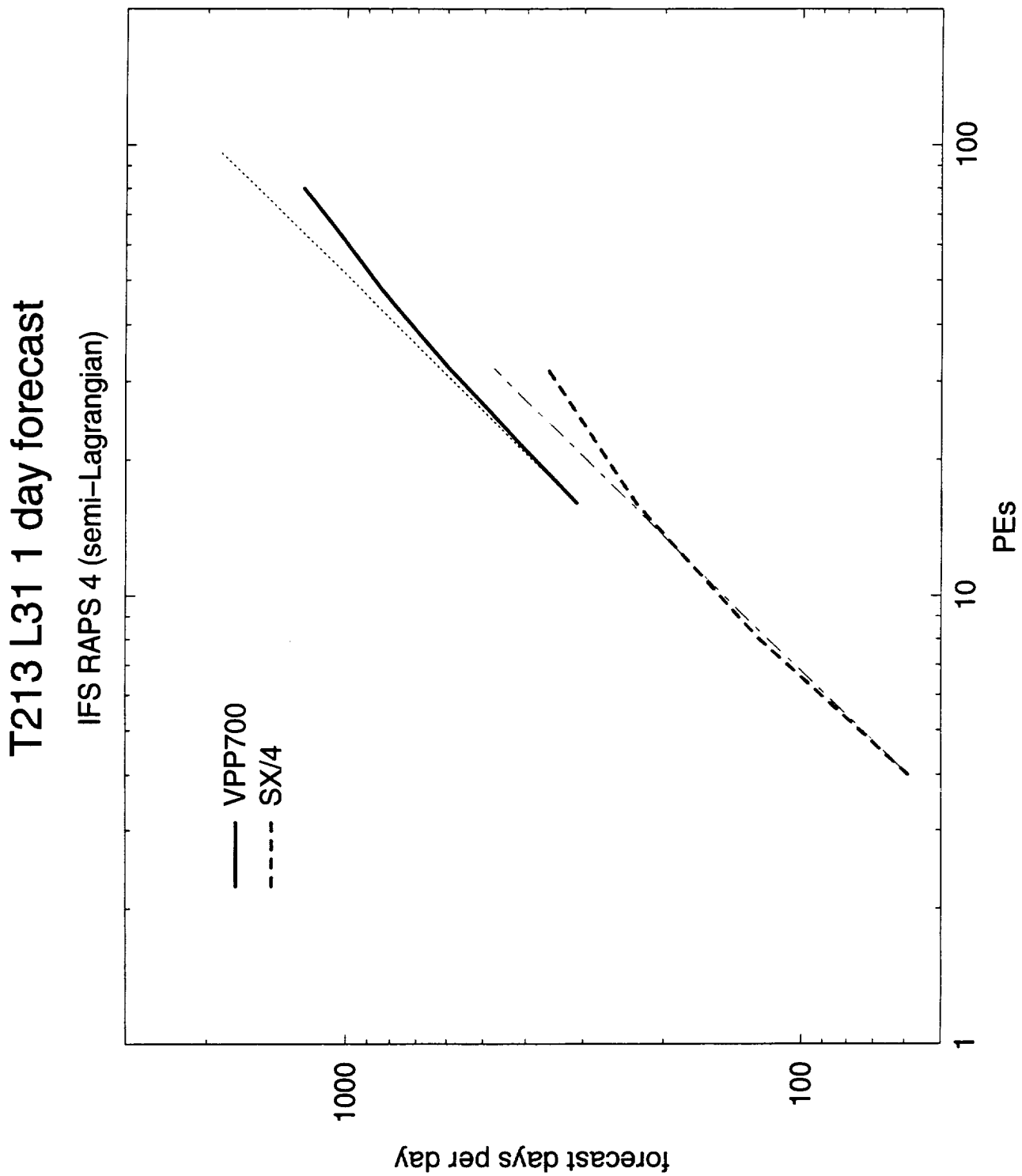


Figure 1: IFS RAPS 4.0 T213L31 Benchmark. Semi-Lagrangian.

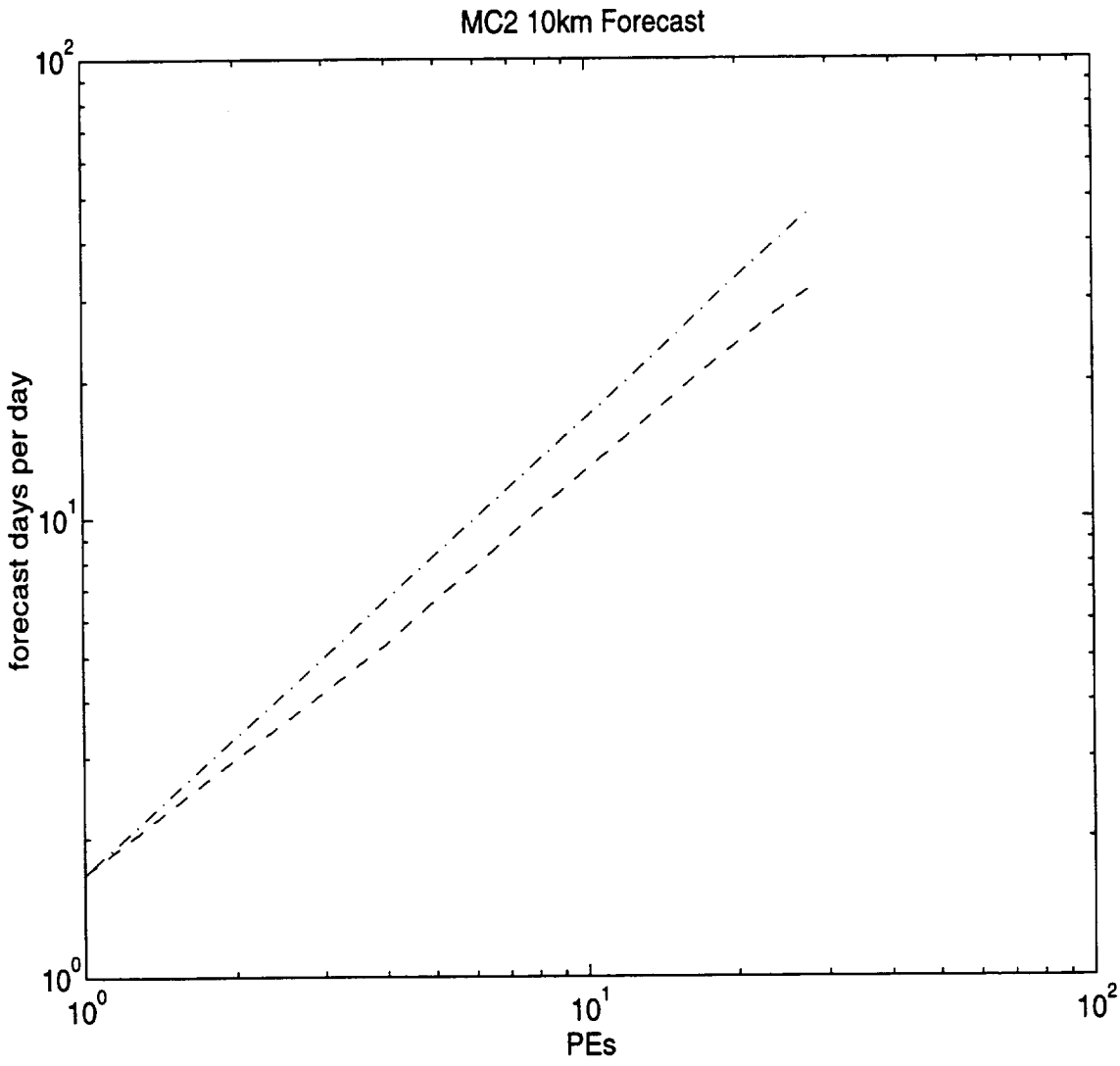


Figure 2: MC2 Performance on SX-4M: Ideal versus observed.

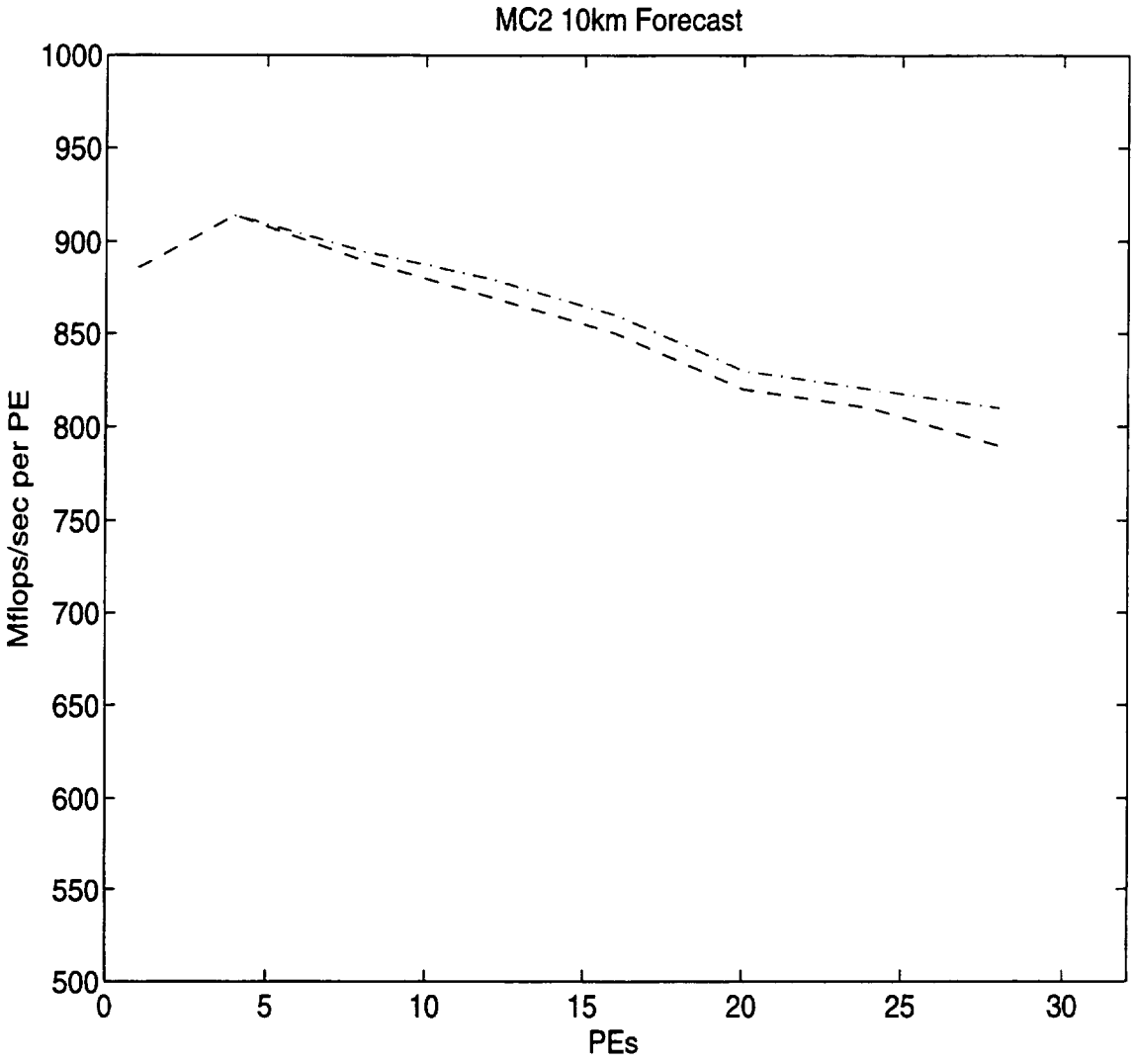


Figure 3: MC2 Performance on SX-4M: Single (bottom) versus multi-node (top).

Author: George Mozdzynski
European Center for Medium-Range Weather Forecasts
Shinfield Park
Reading, Berkshire RG2 9AX United Kingdom
George.Mozdzynski@ecmwf.int

Co-author(s): David Dent
Mats Hamrud
Lars Isaksen

Parallelization of the ECMWF Integrated Forecasting System

The Integrated Forecasting System (IFS) at the European Centre for Medium-Range Weather Forecasts (ECMWF) consists of a large suite of software used primarily to produce a daily 10 day forecast. Key components of the IFS include the processing of observations, a 4 dimensional variational assimilation scheme, and a 10 day forecast model integration using a ~ 60km grid.

In the past, the parallelization of the IFS has concentrated on the forecast model component, and today this continues to show near ideal scaling on many of the parallel computer systems available today. This model has recently been enhanced to support parallel I/O by using some new facilities provided in the MPI2 standard. Results of scalability experiments using these MPI-IO facilities are compared with a parallel I/O package developed at ECMWF.

The processing of observations is an integral part of meteorological data assimilation. The volume of such observational data is growing rapidly with the availability of new satellite data, and requires parallel processing to keep run times to an acceptable level. Observations are stored in a highly compressed BUFR format, which is the WMO standard format for the interchange of observation data. The encoding and decoding of BUFR format files is inherently scalar in nature and does not benefit from vectorization. To resolve this a new and more flexible observational data format has been developed at ECMWF called Central Memory Array or CMA format. The objective is to use CMA format where possible and to constrain the use of BUFR format to archive purposes. An overview of how the IFS performs observation processing is described.

In the last year, ECMWF has introduced a new data assimilation system based on the 4D-Variational analysis method. This method involves the use of adjoint and tangent linear forecast models, and minimization of global cost functions based on observational information and background forecast field information. This method is computationally intensive and requires parallel processing in order to meet time critical requirements. Further, issues of load balance arise from the distribution and cost of processing observations. This paper presents an overview of these issues and how they are resolved, together with performance data from the FUJITSU VPP700 system at ECMWF.

Coarse-grain Parallelization of the CCCma Atmospheric GCM on a NEC SX-4

Bertrand Denis

Canadian Centre for Climate Modelling and Analysis

P.O. Box 1700
Victoria, BC
Canada
V8W 2Y2

bdenis@ec.gc.ca
tel: +1-250-363-8245
fax: +1-250-363-8247

Abstract

CCCma operates spectral atmospheric general circulation models on the super computer system at the Canadian Meteorological Centre (CMC) in Dorval, Quebec. This system includes a multi-node NEC SX-4 Symmetric Multiprocessor (SMP). The nodes are made of vector-parallel platform with 4 to 8 GB of central memory and 16 to 32 CMOS-based vector processors, each with a peak performance of 2 Gigaflops. Presently, the NEC SX-4 cluster at CMC has one 32-processor node (SX-4/32) and two 16-processor nodes (SX-4/16).

An older version of the model (GCM2) is used in CCCma's coupled climate model, which includes atmosphere, ocean and dynamical sea-ice components. GCM2 operates at "T32" resolution in the horizontal and has 10 levels in the vertical. A newer version (GCM3) operates at T47 or T63 resolution in the horizontal and has 32 levels in the vertical. GCM3 enjoys a number of improvements over GCM2, including a double transform methodology, optimal topography to reduce Gibbsian ripples, a new deep convection scheme, improved radiation code, a new land surface scheme (CLASS) and optionally, a semi-Lagrangian treatment of tracers.

We will describe how we have parallelized GCM2 and GCM3 and will report on how we took advantage of the existing code design to get satisfactory performance with the least effort in the context of a coarse grain parallelization on an SMP machine. Diverse strategies concerning the domain decomposition, memory allocation, spectral transform, load balance, and synchronizations will be presented.

1 Introduction

One of the major roles of the CCCma is to help Canadian policy-makers by providing scientifically-based climate simulations (Boer et al.[1], see also the CCCma web site for more information on the centre: *http://www.cccma.bc.ec.gc.ca)*. To pursue this successfully, the main component of the coupled climate model, namely the Atmospheric General Circulation Model (AGCM) has been improved constantly since it was originally developed 20 years ago.

The insatiable need for higher spatial and time resolution, and the increasing complexity of the physical processes represented in the model have continuously pushed the demand for more computing power. To cope with that, the model code architecture has had to be optimized many times in the past. For instance, in the early years of development, main memory was a scarce resource so strategies such as domain decomposition, which allowed partial processing of the computational grid in memory, were adopted. Later, with the acquisition of a vector machine (CRAY-1) the model code had to be vectorized and the vector lengths became another constraint to achieve high performance. Nowadays, the availability of a Symmetric Multiprocessor machine (NEC SX-4) calls for code parallelization to meet a realistic turnaround time for the next generation of climate simulations.

When planning climate simulations for production mode, some factors need to be payed attention to and dealt with such as:

- The available computer resources consisting of, cpu time, central memory, disk memory and archival storage space.
- Spatial and time resolutions at which the simulation is performed. Larger grid point density and the smaller timestep (i.e. more frequent computations) theoretically lead to better results.
- Deadlines for submitting simulated data, for example, to meet international commitments.
- Length of the coupled model simulation(s), typically 200 years or more.

At CCCma, we aim for fully coupled simulations that can be produced in 3 months. Simulations typically cover 2 centuries. This requires roughly 2 years of simulation per real day or 2 months/real hour. For coupled simulations, the work includes the AGCM, the Ocean General Circulation Model (OGCM), the coupling mechanism and data dumping to a mass storage system. The extensive I/O bound diagnostics of the generated climate data are performed concurrently on the front-end (SGI Origin 2000) server.

The first Canadian fully coupled model (Flato et al.[2]) uses the second generation AGCM known as GCM2 (see McFarlane et al. [3]). The second fully coupled model will use a new AGCM called GCM3. As with GCM2, GCM3 is a spectral model but its horizontal resolution has been upgraded from T32 to T47 (optionally T63). In the vertical, the number of levels has been increased from 10 to 32. Besides its higher resolution it enjoys a number of improvements over GCM2, including a double transform methodology, optimal topography (Holzer [4]), a new deep convection scheme (Zang and McFarlane[5]), an improved radiation code, a new land surface scheme (CLASS) and optionally, a semi-Lagrangian treatment of tracers.

These improvements come at a cost:

	cpu time/simulated day
GCM2 T32	30 sec
GCM3 T47	186 sec

For serial code on a single processor these numbers are close to the wallclock time. To meet the turnaround requirement for long climate simulations, GCM3 had to be parallelized to make use of the full potential of the computing system at the Canadian Meteorological Centre (CMC) in Dorval, Quebec on which production runs are performed. This system includes a multi-node NEC SX-4 Symmetric Multiprocessor (SMP). The nodes are made of vector-parallel platform with 4 to 8 GB of central memory and 16 to 32 CMOS-based vector processors, each with a peak performance of 2 Gigaflops. At the present stage of the parallelization development reported here, only a maximum of 16 processors of a single 32-processor node (NEC-SX/32) has been used.

2 Implementation strategies

It was decided to use the NEC multi-tasking capabilities [6] to implement parallelism in the code. We chose GCM2 to start with because it was the production model at that time and any speedup would have been immediately beneficial. Also, the GCM2 code design was much simpler than the developmental model GCM3. The CCCma AGCMs are typical Eulerian spectral models. However, even though we call them spectral models, more than 97% of the work is done in the physical domain, i.e. on a grid. The advantage of the spectral methodology is that horizontal derivatives are computed exactly in spectral space, leaving the computations column-independent on the dynamics grid and avoiding the need for communication between columns. The physics computations are also done in a column-independent manner. Therefore, it was obvious to parallelize first on the computation grids, and then if needed on the remaining 3% which includes the linear spectral computations, I/O, time filtering, etc..

The task of parallelization of a model that has evolved over 20 years without major rewriting was not trivial at first. Fortunately, because early programmers had designed clever

coding strategies to deal with very limited central memory, parts of the model code were already in an ideal state for parallelization. One these strategies was to use a latitude loop to process one latitude at a time instead of loading and computing the full grid all at once. This methodology is also called memory window management. The obvious thing to do first was to take advantage of this memory window management for the distribution of work among the processor elements (PE).

Once we decided that coarse-grain parallelization could be easily implemented in the existing code (which already had domain decomposition built-in), we analyzed the code to locate where synchronization had to be imposed to get reproducibility for any number of PEs given a domain decomposition. We found 2 places that needed attention, both having to do with global summations. The first one involved the total precipitable water and the total tracer amount. The second was in the last operation of the Legendre transform from grid to spectral space. We shall specifically stress this last point in the next section.

One of the targets we focused on when we began the implementation was to get bit-by-bit reproducibility between the original code and the revised one. The researchers were not willing to continue ongoing runs with a multi-tasked model which would not give the same answer. Furthermore, a multi-tasked version which would give the exact same answer would be the proof that the programmer had not introduced flaws. Researchers were more willing to accept different answers for the next frozen version (GCM3) if that allowed for a better speedup. They just asked that the multi-tasked GCM3 yield the same climate from a statistical point of view compared to the original version. To get reproducibility, it was important to identify the memory space (COMMON blocks) that must be private to each task so they do not overwrite each other.

GCM2

The GCM2 was frozen in 1992, but a major optimization to increase the vector lengths for better efficiency has been implemented since. This was done by chaining alternating north and south hemispheric latitudes and made use of symmetrical properties to speedup the Legendre transform. A benefit of this north-south coupling from a multi-tasking point of view is that it provides a first order load balance among the chained latitude sets. In effect, this mixing of latitudes from both hemispheres gives chunks of work which contain physical processes from various parts of the globe. Another advantage of this latitude chaining mechanism is the possibility of getting an exact multiple of the optimum vector length for the NEC-SX4 vector registers which hold 256 eight-byte elements. Figure 1 shows the domain for a 96x48 grid points decomposed in 12 subdomains for 6 PEs.

Figure 2 shows the so-called latitude loop for the original (sequential) GCM2. If using the domain decomposition of Figure 1, the iteration is done sequentially 6 times and just 1/6 of the grid is needed in memory. Figure 3 shows how the concurrency can take place in

that loop which, as said previously, represents more than 97% of total cpu time. Essentially everything is done in parallel with one major exception being the Gaussian quadrature in the forward Legendre transform. Fortunately this step is one of the last ones of the forward spectral transform and explicit synchronization using NEC multi-tasking functions can be used to maintain the order of the summation of the Legendre transform. The bad side effect of this imposed order is an implicit sequential section in the computational flow. The management of the memory for GCM2 was handled by using NEC f77sx compiler directives (*PDIR TASKLOCAL(/xxx/) on COMMON blocks. These were easy to implement once the locality vs. the globality of the data was determined.

Concerning the issue of giving the exact bit-by-bit same answer for the GCM2 version, we first re-compiled with the multi-tasking switch turned on but without any changes in the code. To our big surprise the answer was not the same. It appeared that the code optimization of one expression was not done anymore and thus a one bit difference showed up after 60 timesteps. We finally pinpointed the problem and optimized the code by hand. Then the same answer was obtained and we were set to start the multi-tasking implementation.

GCM3

Apart from its improved physics and higher resolution, the GCM3 computational flow is fundamentally different from GCM2. The new GCM3 make uses of a process-splitting marching scheme where the physics is applied as a correction to the updated dynamical variables at each timestep. In GCM2, the total tendency terms were made of the non-linear dynamics terms added to the physics tendencies lagged by one timestep. Moreover, these physics and the non-linear dynamics tendencies were computed on the same grid. With GCM3, they are performed on different grids having different sizes. This is what we call the double transform. The consequence for the implementation of the parallelization is that, 2 latitude loops instead of 1 must be parallelized, each having a potentially different domain decomposition (Fig. 4).

The explicit synchronization in GCM2 for the Legendre transform was removed by providing the tasks private memory space to put their contributions to the global spectral fields, the final summation now being done after each parallel latitude loop. Again here, the clever implementation of the latitude loop many years ago helps to implement that algorithm. In effect, the forward Legendre transform was designed to be executed almost totally within a latitude set without elements needed from the other latitudes on the transform grid. The individual contributions are only required for the final summation. This was done incrementally as each latitude set is processed in the latitude loop. In other words, using this method, no transpose is needed in the Fourier domain. Mathematically the forward spectral transform is written as,

$$A_n^m = \sum_{j=1}^{NLAT} W(\mu_j) P_n^m(\mu_j) \left\{ \frac{1}{NLON} \sum_{i=0}^{NLON-1} A(\lambda_i, \mu_j) e^{-Im\lambda_i} \right\} \qquad (1)$$

Where $A(\lambda_i, \mu_j)$ are the grid point values, $P_n^m(\mu_j)$ is the (normalized) associated Legendre function and $W(\mu_j)$ are the Gaussian weights, and *i* and *j* are the longitude and latitude indices respectively. NLON is the total number of longitudes and NLAT the total number of longitudes.

The transform is usually performed in 2 steps

1-Forward Fourier transform: (grid->Fourier coef)

$$A^m(\mu_j) = \frac{1}{NLON} \sum_{i=0}^{NLON-1} A(\lambda_i, \mu_j) e^{-Im\lambda_i} \tag{2}$$

2-Forward Legendre transform: (Fourier coef-> spectral coef)

$$A_n^m = \sum_{j=1}^{NLAT} W(\mu_j) P_n^m(\mu_j) A^m(\mu_j) \tag{3}$$

This can be re-written with partial sums $S_n^m(j)$: $A_n^m = \sum_{j=1}^{NLAT} S_n^m(j)$ (4)

where: $S_n^m(j) = \frac{1}{NLON} \sum_{i=0}^{NLON-1} W(\mu_j) P_n^m(\mu_j) A(\lambda_i, \mu_j) e^{-Im\lambda_i}$ (5)

The advantage of this is that each PE can work independently on its own latitude circle to fill up its own spectral triangle $S_n^m(j)$.

When working on a set of chained latitude circles like in Fig. 1, (4) and (5) become:

$$A_n^m = \sum_{k=1}^{NSET} S_n^m(k) \tag{6}$$

$$S_n^m(k) = \sum_{j \in set(k)} \left\{ \frac{1}{NLON} \sum_{i=0}^{NLON-1} W(\mu_j) P_n^m(\mu_j) A(\lambda_i, \mu_j) e^{-Im\lambda_i} \right\} \tag{7}$$

Where $S_n^m(k)$ represents the partial sum for a given set k. When running in parallel, each PE is associated with a given k and evaluates (7) independently. The final summation of each contribution is done serially after the parallel loop.

This method is described in Worley and Drake [7,8] as too expensive in terms of memory, because of the duplication of the spectral arrays, and in terms of communication time on a distributed memory machine. We found that for a shared memory machine like the NEC-SX4 and with the use of dynamic memory allocation these arguments fade away. See also [9] and [10] for more information on parallel algorithm for spectral transform.

With the double transform, the first parallel section of the code, namely the non-linear dynamics, induced no load imbalance. On the other hand, the physics can still produce a load imbalance especially between the PE working at the equator and the one working at the poles since very different physical processes take place in these regions, e.g. convection.

Having now no explicit synchronization in the latitude loop and because much more work is done in the more expensive GCM3 physics, 98.5% of the total work in this version can potentially be executed in parallel. But because of some physics load imbalance and because some serial time is needed at the beginning of each latitude loop to obtain the PEs and to get them working on their task, the effective amount of work done in parallel is about 97% for T47 when using 6 PEs.

For GCM3, we still use the compiler directives (*PDIR TASKLOCAL(/xxx/)) for local common blocks but we moved to dynamic memory allocation using integer pointers for the biggest work arrays. Tests were done showing that the replacement of the compiler directives by the use of dynamic memory allocation yielded up to 25% more speedup. In addition, when the tasks were generated for a given latitude loop, the memory for all declared tasklocal commons in the code was allocated on the stack, not just the memory used in that latitude loop.

3 Performance results

All performance tests were done on the operational node which is used not only for model development and climate production runs, but also by the Canadian Meteorological Centre (CMC) for weather forecast production. This node has 32 PEs available. Considering the other users on the machine and the fact that we had climate production runs, 6 to 16 PEs were used. The results are for one model simulated month.

GCM2

For GCM2, the spectral truncation is T32, the physics/dynamics grid is 96x48, the number of vertical levels is 10 and the timestep is 20 minutes. Table 1 gives a summary of the performance as a function of the number of PEs. A speedup of 4.2 is obtained with 6 PEs.

GCM3

For GCM3 two sets of tests are shown. The first is at T47 with physics grid of 96x48 and dynamics grid of 144x72. The second is at T63 with a physics grid of 128x64 and a dynamics grid of 192x96. The timesteps are respectively 20 and 15 minutes. Both have 32 vertical levels. Tables 2 and 3 summarize performance for these 2 resolutions. At T47 and with 6 PEs, the execution time is 18 minutes which represents a speedup or 5.2 . At T63 and with 16 PEs, the execution time is 22 minutes and the speedup 10.2 .

In the above tests, the shortwave (sw) radiation computations were done every hour. When they were done at each timestep, the T47-6 PEs version ran at 4 Gflops taking 26 minutes per month and the T63-16 PEs version ran at 8.5 Gflops taking 27.7 minutes per month. Considering that the climate with the sw computed at each hour is very similar to the one computed at each timestep, it has been decided to go with the less expensive alternative, i.e., once an hour.

Conclusion

Substantial speedups have been achieved without rewriting the entire AGCM code. This will help us to perform our long term climate simulations within the existing constraints described in the introduction. In effect, the wallclock time for GCM3 T47 decreases from 186 sec/day to 36 sec/day using 6 PEs. With this performance, the goal of 2 simulated months per hour is easily met. At T63, 16 PEs, the same time target is also met. However, in a coupled mode, the OGCM also requires a substantial computational effort. We are about to start multi-tasking the OGCM code and are developing a flux coupler using MPI communication capabilities.

Speedups shown in table 2 and table 3 do not scale very well for a relatively large number of PEs. Considering that a single node SX4 has a maximum of 32 PEs and that we usually run at least 2 concurrent production runs on it, this kind of scalability is acceptable. Anyway, a speedup of 10 for GCM3 at T63 is something that makes model users very happy!

Acknowledgement

The author would like to thanks Fouad Majaess for the long hours of discussion on the project. Special thanks too to the CCCma AGCM model manager, Mike Lazare. Also many thanks to my CCCma colleagues who provided me precious comments and helps.

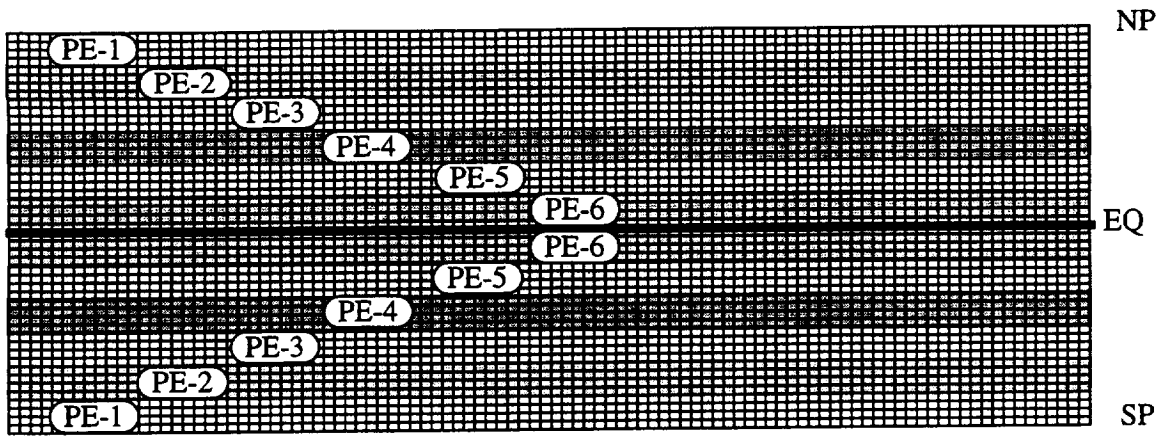


Fig. 1. Domain decomposition of a 96x48 grid for 6 processor elements (PE-1 to PE-6). Each PE shares part of each hemisphere. Each PE performs computations on vectors having optimum multiple length of 256 elements. This is done by chaining alternatively north and south a total of 8 latitude circles of 96 grid points each.

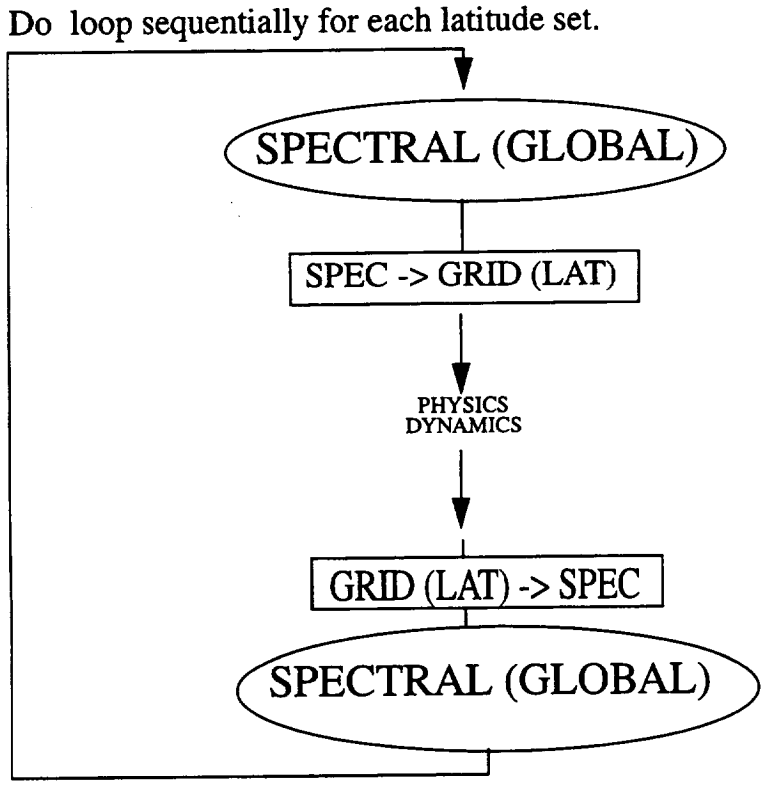


Fig. 2. Original so-called latitude loop for GCM2. The time goes downward beginning with the inverse spectral transform at the top. The forward transform is at the bottom. One latitude set is processed at a time. For the typical 96x48 grid with 8 chained latitudes, the sequence is repeated 6 times.

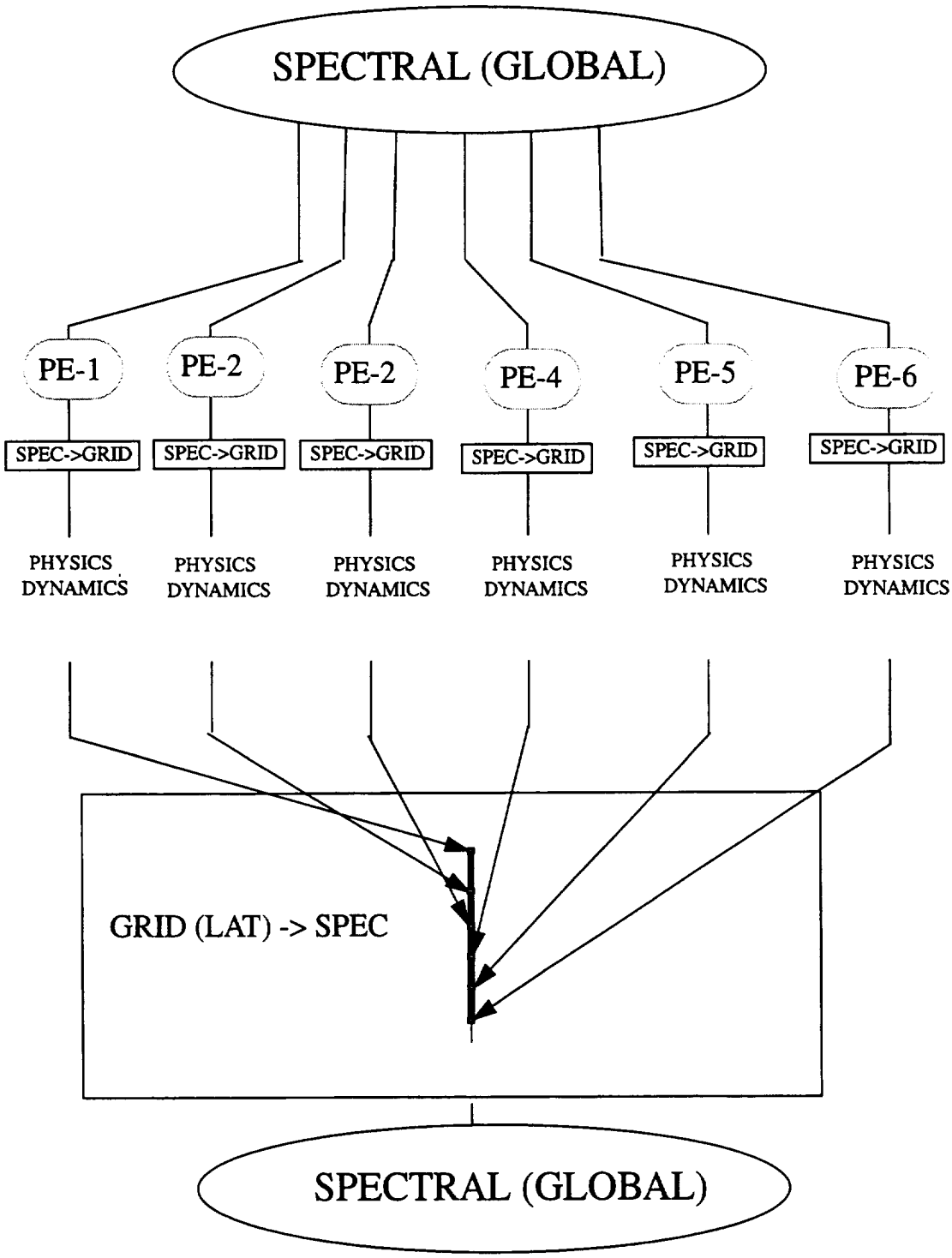


Fig. 3. As in Fig. 2 but with 6 PEs working concurrently on their own latitude set. The heavy line, where the arrows converge, represents the area where explicit synchronization is imposed. Note also that, for GCM2 considered here, the physics and non-linear dynamics are done on the same grid.

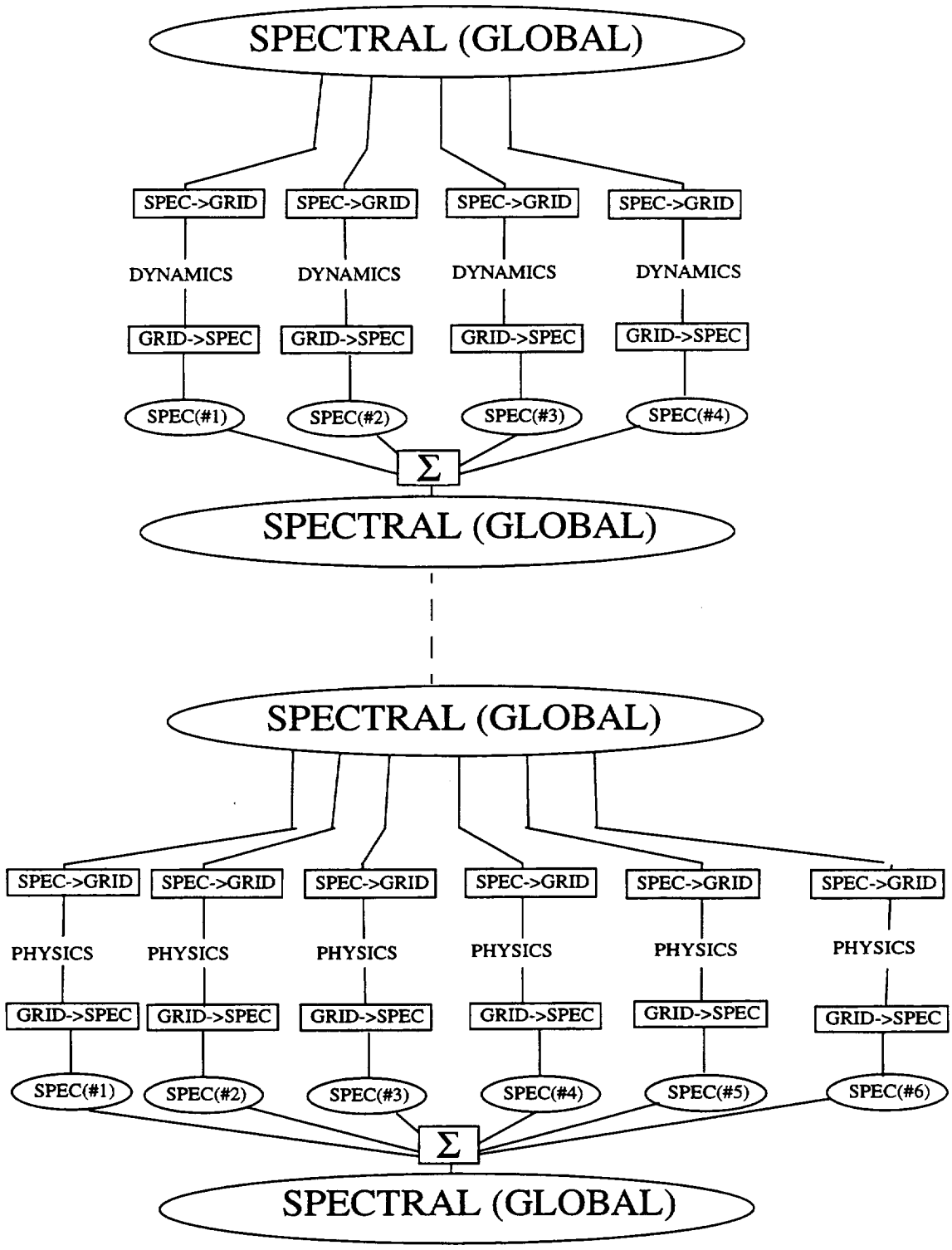


Fig. 4 Computational flow for a GCM3 timestep showing the 2 parallel latitude loops. The first one is for the non-linear dynamics and the second for the physics. SPEC(#) indicates private spectral arrays. This diagram is for 4 PEs on the dynamics grid and 6 PEs on the physics grid.The final summations are done after each corresponding latitude loop so that no explicit synchronization is done inside the parallel latitude loops.

Table 1: Performance for GCM2 T32

P	Mem (mb)	Exec Time (min)	CPU Time (min)	Gflops (total)	Mflops (avg/ proc)	Speedup	Efficiency
1	50	15.3	15	0.530	530	1	
2	68	8.5	15.5	0.955	478	1.8	90%
3	83	6.0	15.7	1.341	447	2.5	84%
6	128	3.7	16	2.215	369	4.2	70%

Table 2: Performance for GCM3 T47

PE	Mem (mb)	Exec Time (min)	CPU Time (min)	Gflops (total)	Mflops (avg/ proc)	Speedup	Efficiency
1	180	93	93	0.71	710	1	
2	322	49	94	1.35	706	1.9	95%
3	488	34	95	1.95	702	2.8	93%
6	833	18	96	3.62	693	5.2	86%

Table 3: Performance for GCM3 T63

PE	Mem (mb)	Exec Time (min)	CPU Time (min)	Gflops (total)	Mflops (avg/ proc)	Speedup	Efficiency
1	168	224	224	0.66	660	1	
4	441	63	229	2.24	560	3.5	89%
8	830	35	234	4.21	525	6.4	80%
16	1490	22	238	6.68	417	10.2	63%

References

[1] Boer, GJ, McFarlane NA, Lazare M, "Greenhouse gas-induced climate change simulated with the CCC second-generation general circulation model". J. Climate, 5, 1045-1077, 1992.

[2] Flato, G.M., G.J. Boer, W.G. Lee, N.A. McFarlane, D. Ramsden, M.C. Reader, and A.J. Weaver. "The Canadian Centre for Climate Modeling and Analysis Global Coupled Model and its Climate". in manuscript, 1998.

[3] McFarlane NA, Boer GJ, Blanchet J-P, Lazare M, "The Canadian Climate Centre second generation circulation model and its equilibrium climate". J. Climate, 5, 1013-1044 , 1992.

[4] Holzer, M., "Optimal spectral topography and its effect on model climate", J. Climate, 9, 2443-2463 , 1996.

[5] Zhang, G.J., and N.A. McFarlane, "Sensitivity of climate simulations to the parameterization of cumulus convection in the CCC GCM, Atmosphere-Ocean, 33, 4097-446, 1995.

[6] NEC, "SUPER-UX FORTRAN77/SX Multitasking User's Guide", release 7.2 (1997).

[7] Worley H.P. and J.B. Drake, "Parallelizing the Spectral Transform Method", Concurrency: Practice and Experience, Vol.4, p. 270-291, 1992.

[8] Worley H.P. and J.B. Drake., "Parallelizing the Spectral Transform Method. Part II", Concurrency: Practice and Experience, Vol.7 p. 509-531, 1992.

[9] Foster, I.T. and P.H. Worley, "Parallel Algorithms For Spectral Transform Method", Tech. Report ORNL/TM-12507, OakRidge National Laboratory, 1994.

[10] Foster, I.T., B. Toonen and P.H. Worley, "Performance of Parallel Computers For Spectral Atmospheric Models, Tech. Report ORNL/TM-12986, OakRidge National Laboratory, 1995.

KEY ELEMENTS OF THE USER-FRIENDLY, GFDL SKYHI GENERAL CIRCULATION MODEL

Richard S. Hemler

Geophysical Fluid Dynamics Laboratory/NOAA
P.O. Box 308
Princeton University
Princeton, New Jersey 08542
e-mail: rsh@gfdl.gov
tel: +1 609 452-6598
fax: +1 609 987-5063

1. Introduction

SKYHI is a grid-point atmospheric general circulation model which was developed at the NOAA Geophysical Fluid Dynamics Laboratory nearly twenty years ago (Fels *et al.* [1]).The model is global, and extends in the vertical from the ground to about 80 km. SKYHI has been used to investigate various troposheric, stratospheric and mesospheric phenomena, including sudden stratospheric warmings, the quasi-biennial oscillation, ozone depletion, gravity wave-mean flow interactions and chemical and tracer transport problems. A concise description of the model equations, numerics and physics may be found in Jones *et al.* [2]; a more detailed description is found in Hamilton *et al.* [3].

Jones *et al.* [2] describe changes made to the model in order to port it to the Thinking Machines Corporation CM-5 machine at Los Alamos National Laboratory. This paper concentrates on changes made to the model that significantly enhance the ability of the SKYHI user to quickly and easily modify the model code to initiate, run and analyze new scientific experiments, and to attach new features to the model with a minimum of difficulty. Section 2 discusses the development of the generic SKYHI code in which the efficient use of human resources is balanced with the efficient use of machine resources. Section 3 defines the basic structure of SKYHI and the current meaning of "modularity" as used by SKYHI. In section 4, two of the important new user-friendly "modules" of SKYHI are presented, along with examples of their use. Section 5 discusses the parallel performance and scaling of SKYHI on CRAY PVP shared memory and CRAY T3E distributed memory machines, while Section 6 serves as a summary.

2. Balanced optimization

The SKYHI model code that existed in 1990 was the result of many people's efforts over many years and several machines to produce a code which would run using a minimum amount of CP time, memory and I /O time. As a result, the relationship between the model code and the analytical expressions it represents was difficult to see; model variables were often defined to minimize arithmetic operations, rather than to be physically- or numerically-meaningful quantities. Memory usage was minimized by implicit and explicit equivalencing; certain common blocks were used to provide the current functionality of the stack. Specific coding practices needed for performance on previous platforms were still in place, even though they were no longer needed. The result was a model which could be run with scientifically-meaningful horizontal resolution on a machine with limited central memory, but which was difficult to modify and often failed in unexpected ways when disturbed, and in which code for specific model processes was scattered across many subroutines.

At the same time, the shrinking budget for basic research and the desire to reduce the Fed-

eral workforce was shrinking the number of GFDL programmers and scientists available to manage the code and to run scientific experiments. To take up the slack, more visiting scientists were being invited to GFDL, usually for periods of a few years. In order for these people to make productive use of their time at GFDL, it was essential that they quickly become able to use the model in their studies, and not spend a lot of time attempting to attach their experiment to SKYHI. The existing code structure of SKYHI made this process difficult.

With the coming of the CRI YMP machine in 1990, the available computing power at GFDL increased significantly. This allowed the "machine de-optimization" and "user optimization" of the SKYHI code to begin, without resulting in a reduction in model throughput for the SKYHI user community. Code constructs which may have been machine efficient at some point in the past but which were difficult for users to understand and modify correctly were replaced with more easily understood code, the first step toward balanced optimization.

Balanced optimization attempts to optimize not only the use of machine resources, but also the human resources required in scientific investigations. Fig. 1 is a schematic representation of some of the important factors that must be considered in a *research* code designed to be used by a variety of users. The ideal code would be optimal in all of these user and machine resource utilizations; since no real code is ideal, it is desirable to produce a code in which none of these features is out-of-balance with the others. Typically these features compete with each other; for example, a code which is highly I/O efficient is likely to be less memory efficient than it could be if the I/O were not so efficient. The strategy then is to produce a generic production code in which there exists a balance between these features.

This generic code is the form that is maintained and made available to users. It is written so that it allows easy access to the model to a broad spectrum of users with diverse research interests, each of whom may wish to view different parts of the model as a black box, and who may not need or want all of the features that are provided. Individual users then optimize the generic code as they must in order to reach their scientific goals.

3. The structure of SKYHI and modularity

Figure 2 shows the four basic sections of the SKYHI model. The model's initialization phase consists of calling subroutines to set up tables, initialize constants, read input data -- all the things that are only done once during a model run. After completion, the time-step loop is entered. Some of the calculations in this loop are not a function of gridpoint, but only a function of the time or timestep. This is called the time-dependent phase. After it is executed, the chunk loop is entered. This loop extends over the model domain, which

is broken up into rectangular portions in the horizontal, referred to as chunks. These chunks may be executed either sequentially or in parallel. Within this prognostic section the model equations are time-integrated, contributions from the chunk to any integrands being calculated are computed, and any data from the chunk which are needed in archive files are written. After the domain chunks are processed, the processing of global integrals

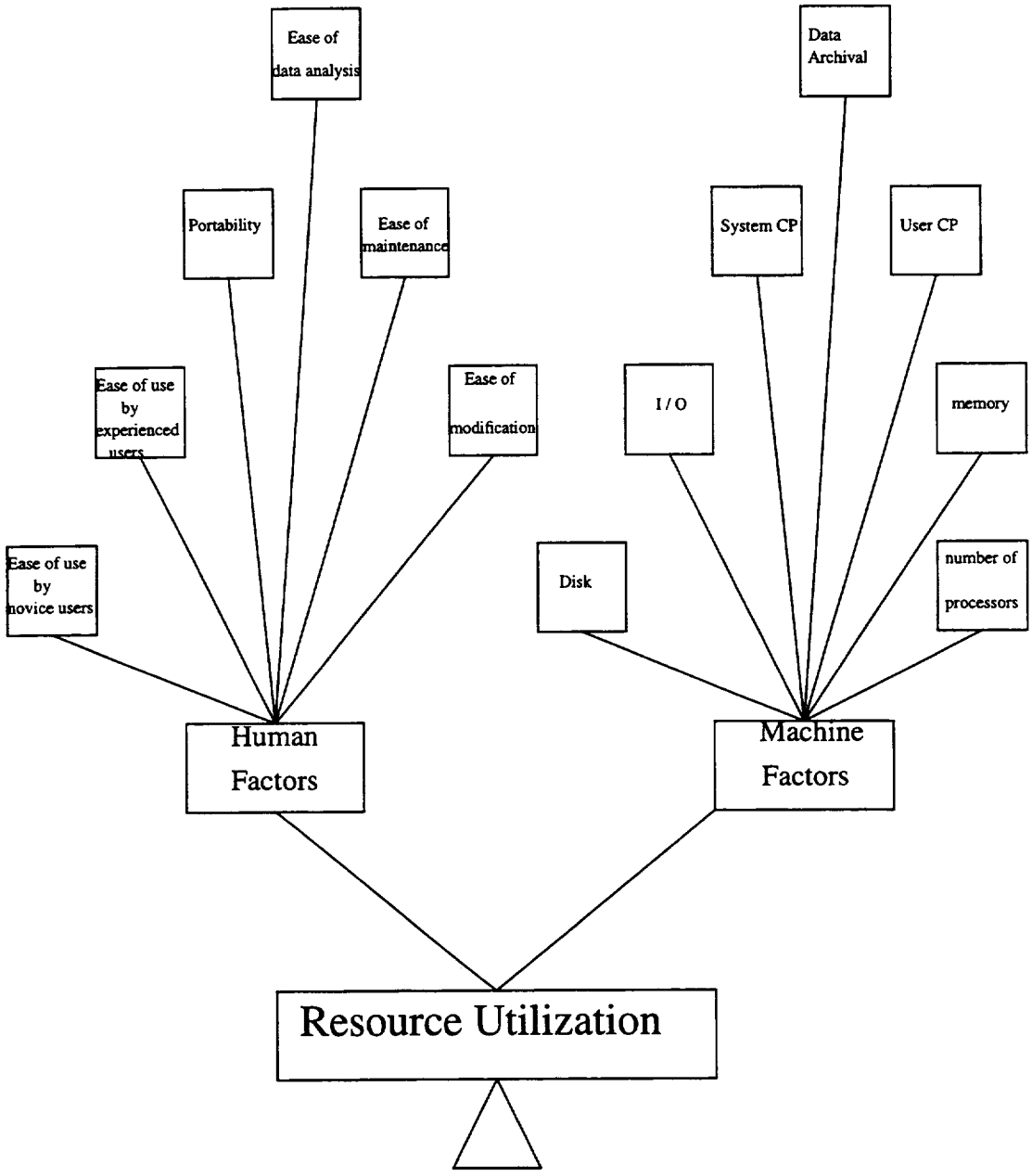


Figure 1. A schematic representation of the human and machine factors which must be considered in balanced optimization.

is done in the diagnostic code section to complete the time step.

Part of the ongoing effort to make SKYHI more user-friendly is the separation of the source code into "model" code and a number of modules. In SKYHI at this time, a module is defined simply as code needed to perform a specific model function which has been isolated from the rest of the source. That part of the code not yet contained within a module is referred to as the "model". The following guidelines must be met for a portion of code to be called a module in SKYHI:

1) No "model" include files, common blocks or subroutines may appear in any module.
2) No module include files, common blocks or subroutines may appear in the "model".
3) All communication between "model" and a module must be through argument lists.
4) Module to module communication must occur by going through the "model".
5) Modules may have an interface to the "model" in each model section defined above.

These conditions assure that the communication between model and module is explicitly defined, via a subroutine call argument list. It makes it obvious to a user what model variables must be supplied to the module, and prevents the inadvertent modification of model variables that could be brought into a module via a model include file or common block. Experience has indicated that many argument lists may be shortened by some code reorganization, which also invariably produces a more understandable and more modular code.

At the moment, SKYHI contains several physics modules that follow these coding guidelines. These include a radiation package, a cloud package, a surface albedo package, an astronomy package and an ozone package. These modules have been carved out of the previous SKYHI radiation code by isolating the code which is related to these processes. A modular structure allows easy implementation of alternative parameterizations and also allows the output from these modules to be passed back to the model and then used as input to other parameterizations, e.g., the stratospheric chemistry package associated with SKYHI requires astronomy package output. More processes will be pulled out of the "model" and made into "modules" as time permits.

Several modules have been added to SKYHI recently, following the coding guidelines presented above. These added modules include two optional tracer transport packages (advection plus subgrid-scale diffusion) and a particle trajectory package. An interface for each module was created in each of the four model sections (initialization, time-dependent, prognostic, diagnostic), and the package code which should be executed in each model section is accessed through this interface. As time allows, the modules in SKYHI will be made compliant with the Fortran 90 module construct, further increasing the ease with which new Fortran 90 code written by others may be incorporated into SKYHI.

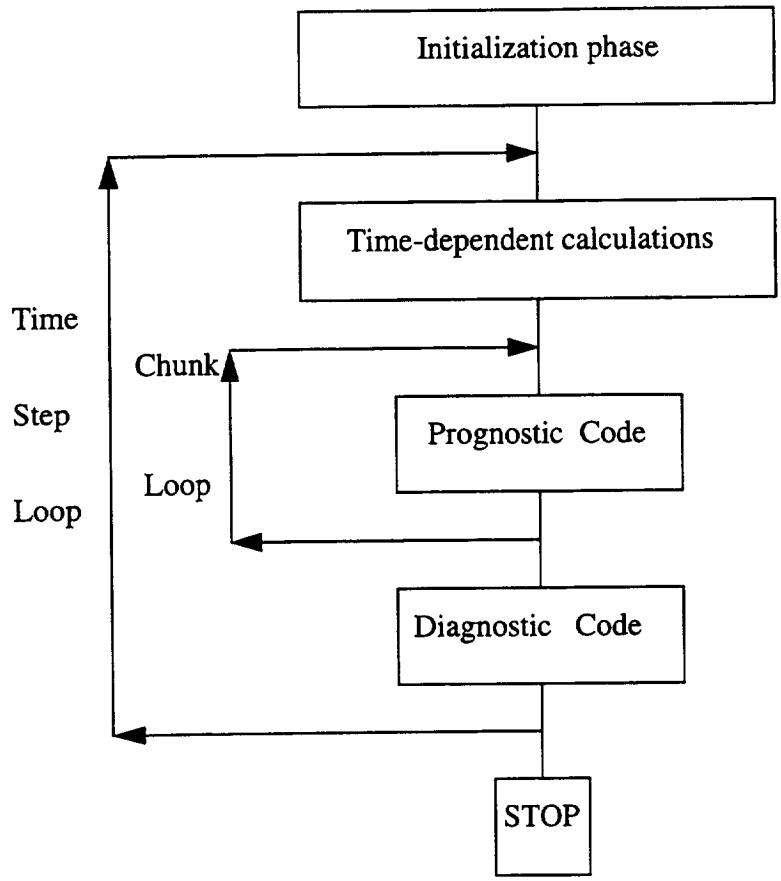


Figure 2. The basic structure of the SKYHI model.

4. The archiving module and the user variable module

Two new higher-level modules have been created in SKYHI. These modules handle two functions which are essential for users who must significantly alter the existing code for their experiments. Both of these features help to isolate the user's code from the SKYHI model code and so reduce the chances of the user inadvertently modifying the model code.

a) The archiving module

The archiving module controls the writing of data files that will be analyzed offline. Five file types are currently recognized: restart, point data, column data, slab data and reduced data. A restart file is written at the end of a job, and must contain the time-dependent variable fields and any other data that are necessary to allow the model to be started again at a

later time, as though it had never been halted. Point data, column data and slab data files contain the desired model variables at a specified set of individual grid points, grid columns or grid planes, respectively. In the limit, the slab data files will cover the entire model domain. Reduced data files contain non-gridded data; they may contain integrals over the globe or some portion of it, or any other grid-independent data set .

All of these files (with the exception of the restart file) may capture either a "snapshot" of the data or may be used to contain some type of time-averaged representation of the data. Standard forms for each of the five file types are provided with the model code as a template to be used in creating customized files. Three standard forms are provided for each file type: the form for the standard version of the file; the form which will produce a time-averaged standard version of the file, and a form which may be customized by the user to provide whatever set of variables is desired, either as a snapshot or as a time-average.

User control of the archive files is provided through a combination of pre-processor options, namelist variables and model parameters. Pre-processor variables define how many file forms of each file type are present in the code. If the user adds code for a new file form, then the variable for that file type would be increased from the default value. If the model is to be integrated without writing any files of a given type, then the variable for that type is set to zero, and all the code and storage associated with the data files of that type will be removed from the model source.

If any data files for a given file form are to be written, then the user must specify namelist variables that define the temporal characteristics for each of the file forms of that file type. These variables and their use are described in Appendix A.

Data for each of the file forms of the grid-dependent file types is collected in an array dimensioned by (i, j, n) where i and j refer to the horizontal grid points in x and y respectively, and n is the sum over all variables of the number of k levels at which each variable is to be written. The beginning and ending spatial location indices of these arrays and the value of n for a given file form are specified as parameters in the model code.

Appendix B defines the recommended procedure for a user to follow when using the archiving module.This procedure has been successfully followed by SKYHI users. The naming conventions and supplied code allow even an inexperienced user to successfully create new archive file forms by making it evident what code changes must be made to the default code to create a new file form. Equivalent variables for different file types have names which differ only by the unique letter associated with that file type. Equivalent variables for different file forms of a given file type have names which differ only by the file form number, which is (are) the last digit(s) in the variable name. For example, the parameter defining the record length of the data record is **I*t*RECLEN*a*** where t is the file type (r,

t, s, h, or i) and ***a*** is the file form number. Thus the variable names are very recognizable, and contain the file type and file form information within them. The creation of control code for new file forms simply requires the duplication of existing supplied code, the location of the file-form-specific variable names, all of which contain the file form number, and the replacement of the old file form number with the new in these variable names.

The archiving package has also been incorporated into the GFDL Limited-Area Non-hydrostatic model (LAN) without difficulty, and should be insertable into any atmospheric model which has the proper interfaces. New and useful Fortran 90 constructs will soon be incorporated into the package and any problems or lack of clarity reported by users will continue to be addressed. Additional capabilities are also planned, including the ability to create time-averaged files that span jobs.

b) The user variable module

The user variable module allows the user to easily add variables to the model. The model contains an array ooo (*ist:iend, jst:jend, kst:kend, n, m*), where the first three dimensions are the (*i,j,k*) spatial indices, *n* is the number of ooo variables, and *m* is the time level index, indicating lag, mid or lead time. This variable is available to the user to contain *n* variables of his choosing. If no user supplied variables are desired, then a preprocessor option is set, and all the code and storage associated with this variable is removed from the code. When user-defined variables are desired, model-supplied code will handle all aspects of their integration except for the calculation of the specific physics-chemistry-source-sink terms relevant to the variable in question, which obviously must be supplied by the user.

These user variables may be fully prognostic, semi-prognostic, or diagnostic. Fully prognostic variables have a time tendency resulting from transport and may also contain physics-chemistry-source-sink terms. The transport is handled by the model, as specified by the user from a series of options that are offered; there is no need for the user to provide code to transport these variables, unless a scheme is desired that is not offered by the model. Semi-prognostic variables have a time tendency resulting from source / sink terms, but are not transported. Diagnostic variables are defined on the basis of other conditions, and do not have an explicit time tendency equation. These variables may be integrated with a timestep either smaller or larger than the model timestep, if desired.

The user must customize the general user variables so that they become the variables that are desired. The model contains the chemical species nitrous oxide as a sample ooo variable, the treatment of which the user can follow for his own variables. Interfaces between the model and user variable module are provided in each model section; the user must determine the variables that will become the argument list between model and module.

Different types of variables have been coupled with SKYHI using the user variable code block. The experience gained in coupling these diverse types of variables to SKYHI has produced a more general structure of the user variable block, so that all of these variable types, each with their own special requirements, may be handled properly. The result is a more robust module, one that is more likely to be able to handle the next set of variables thrown at it than it was previously.

Several different uses have been made of the user variables in a chemistry context. The investigation of tracer transport by different transport schemes has been done very neatly by setting up an experiment with several initially identical copies of a given species, each of which is integrated with a different transport scheme, all within the same model run. The data for all the schemes are then present in the same data files, simplifying the analysis effort. A stratospheric chemistry package with thirty-seven chemical species has been attached to SKYHI and run without difficulty. In this case, some variables are prognostic, some semi-prognostic, some diagnostic, and some may be prognostic or diagnostic, dependent on the time of day. The option to have variables change between prognostic and diagnostic was not originally present and required a generalization to the original code, which was successfully done.

Experiments that have investigated the vertical diffusion scheme in SKYHI have employed a simple radon tracer with a surface source. Multiple versions of the diffusion scheme may be tested in the same job by starting multiple identical copies of the tracer, each as a different user tracer variable which has a different diffusion scheme. In such a case the model needs to be run only once in order to compare n different diffusion formulations, rather than n times.

Another use of the user variables has been with a cloud ice parameterization scheme. Here there was a need for twenty-three diagnostic variables, and the presence of the user variable block allowed the easy inclusion of that many new variables. It is anticipated that cloud microphysics and atmospheric aerosols may soon be examined in SKYHI and it is likely that both of these variable sets will be handled within the user variable block.

The user variable module will also provide the flexibility needed to handle alternative column physics packages that are available as options within the model. Different parameterizations of a given process will require different variables to be communicated between the model and the package, and so a different interface could be needed for each package. However, using the user variable array ooo will allow each package to communicate with the model through the same interface, thus simplifying the source code.

Section 5. Performance on parallel systems

One of the optimization factors shown in Fig. 1 is the ability for code to run in production mode on multiple processors. This feature is essential to avoid limiting the numerical experiments which may be undertaken and the platforms upon which the model can be run.

Two major changes were necessary to allow the code, which had been running on a single processor, to be run on multiple CPUs of the Cray PVP machine. The original unitasked code executed the model one latitude row at a time (the chunk loop of Fig. 2) because of memory constraints, marching from south to north, providing a natural coarse-grained, one-dimensional data decomposition scheme for parallel execution. In unitasked mode, this decomposition allowed the center and northern row variable fields and the fluxes that had been calculated at the northern boundary of a grid row to be saved and used as the southern and center row variable fields and southern boundary flux of the next row. Thus each new row required only the reading of the new northern row of data from disk and the calculation of the northern boundary fluxes. However,with multitasked execution, each processor must calculate fluxes at both boundaries, and read all the data it needs from disc, since it is not certain that it will have the data from the previous row. These two changes result in a 10-15% increase in CP time for SKYHI, but allow the model to be run on parallel systems.

The scaling performance of SKYHI on the GFDL CRI T932 machine during a dedicated test time is summarized in Table 1 for both one degree latitude (N90) and for one-third degree latitude (N270) resolution. These numbers were obtained by running the model for several timesteps without archiving any data, and do not include the time spent in the initialization section of the model, which is primarily spent reading the initial data and is not parallelized. It is seen that scaling for both resolutions deteriorates above 9 processors. This decay reflects the single-threaded nature of access to the model data stored on the SSD, meaning that as the number of processors increases, processors must wait longer to get the data they need. A further degradation of performance occurs on 24 CPUs, which presents an inherent load balancing problem for 180 or 540 chunks.

Thus, on the Cray PVP machine, a one-dimensional domain decomposition is adequate, since the number of processors which can be efficiently used on a problem will be limited by the single-threaded SSD access time before the load balancing problem resulting from the limited number of latitude rows becomes important. Currently only the one-third degree latitude SKYHI experiment is being run multitasked in production mode on the GFDL T932; lower resolution runs are run unitasked to take advantage of the savings discussed above. The one-third degree experiment is multitasked 4 ways; on 4 processors the code is still parallel efficient and the machine resource usage is balanced. The model at this resolution requires 12% of the total system memory and 15% of the total system SSD,

so that 4 processors (15% of the total of 26) is reasonable.

When SKYHI was ported to the LANL TMC CM-5 machine (Jones *et al.* [2]), however, it was necessary to use a two-dimensional domain decomposition. The major longitudinal dependence which was encountered was the polar Fourier filter, which required extensive communication of data between processors, and resulted in a high computational price. Jones *et al.* [2] discuss the performance of SKYHI on the CM5 machine and the reasons for it.

Table 1: SKYHI PVP scaling characteristics

Resolution	Number of processors	Wall clock time (seconds)	Scaling	Parallel Efficiency
N90	1 (unitasked)	334	-	-
N90	1	369	1.00	1.000
N90	3	125	2.95	0.983
N90	9	44	8.39	0.932
N90	18	26	14.19	0.783
N90	24	24	15.38	0.641
N270	1 (unitasked)	267	-	-
N270	1	308	1.00	1.000
N270	3	104	2.96	0.987
N270	9	35	8.80	0.978
N270	18	23	13.39	0.744
N270	24	23	13.39	0.558

The SKYHI code running on the Cray PVP machine has also been modified to run on the CRI T3E distributed memory machine. Pre-processor options are used to select either the shared memory or distributed memory version of the code. These code versions differ by about 1200 lines, primarily in code involving the T3E domain decomposition and assignment of data to processors, the storage of tau file data in local memory rather than on disc, communication of data between processors (shmem calls), and the data archiving process. The code remains that which has been optimized for the vector machine; no specific T3E

optimizations have been made. Raw performance of this code on the T3E is about 36 Mflops/pe, compared to 500 Mflops on a single T90 CPU.

The major source of load imbalance in SKYHI is the polar Fourier filter. All latitude rows which are filtered take about the same time to execute, as do latitude rows which are not filtered, with the difference in time between filtered and non-filtered rows being about 15%. This information may be used to decide how to assign latitude rows to processors, and so better balance the load across processors, in contrast to a simple round-robin assignment of rows to processors. For example, the best balanced load for an experiment with 180 latitudes was obtained using eight processors (in contrast to nine, ten or twelve), even though the eight processors were not all responsible for the same number of latitude rows. However, as the number of processors approaches the number of latitude rows, the ability to balance the load decreases, and so makes the round-robin approach as good as any. It is also of some advantage to assign contiguous latitudes to the same processor, all other factors being equal, and so reduce off-processor communication.

Scaling results for a one-degree latitude version of SKYHI with 160 vertical levels run on the 512 processor T3E at the National Energy Research Scientific Computing Center (NERSC) are shown in Table 2. As in Table 1, these results are from several timesteps of integration, without archiving data, and do not include the time spent in the initialization section of the model. This resolution requires a minimum of fifteen 32-Mw T3E processors in order to have enough memory per pe to integrate the model. It cannot be integrated on the GFDL forty 16-Mw processor T3E system with one-dimensional domain decomposition; there is not sufficient memory per processor to contain the data needed for one latitude row. The degradation of performance with increasing number of processors seen here is relatively less than on the T90, since SSD access bottle-necks are not involved. Instead what is seen is the reduction in the ability to balance the load as the number of processors approaches the number of chunks of parallel work (180).

Table 2: SKYHI T3E scaling characteristics

Number of processors	Wallclock Time (seconds)	Scaling	Parallel Efficiency
15	266	15.0	1.000
30	135	29.55	0.985
45	93	42.9	0.953
60	72	55.42	0.924

Section 6. Conclusions

The increase in the relative value of human resources compared to machine resources at GFDL in recent years means that the definition of code optimization must be changed to include human factors in addition to the traditional machine resource usage. A meteorological model used in research must be structured so that investigators not familiar with the details and history of the model may quickly learn enough about it in order to use it productively in their scientific research. This user-friendliness will usually come at the expense of machine performance, a condition which must be accepted in order to optimize the total scientific productivity of the model and of the scientists who use it.

A user-friendly model requires at a minimum that the code is "modular", meaning that the different model processes communicate with the rest of the model in clearly specified ways, as opposed to being intertwined. In this way investigators may easily examine individual parts of the model, without having to extract the process of interest from a dense ball of code, a process which often proves to be both difficult and time-consuming.

The restructured GFDL SKYHI general circulation model has also addressed two specific topics which in the past have inhibited investigators in their productive use of SKYHI; the ability to easily define new data files for later off-line analysis of the model output, and the ability to easily add new variables to the model for specific investigations. The archive module and user variable module described here have a standard format and are flexible to user needs. These packages will continue to evolve in response to user desires and complaints, becoming more user-friendly over time. Inclusion of Fortran 90 constructs in these packages should result in some performance improvements and ultimately cleaner code, albeit code which will look less familiar to the current user community.

SKYHI has been successfully integrated in production mode on the Cray T932 PVP machine in both unitasked and multitasked modes using one-dimensional domain decomposition. Parallel performance scales well with number of processors (if obvious load imbalance configurations are avoided) to the point where the single-threaded nature of SSD access limits performance. The same PVP-friendly source has been successfully run in production mode on the CRI T3E machine using one-dimensional decomposition. Parallel performance on the T3E is relatively better because of the absence of the SSD bottleneck, but is ultimately limited by the one-dimensional domain decomposition.

At this time, three major areas remain which are negatively impacting the performance of SKYHI on parallel systems. Single-processor performance remains an issue; whether improvements in cache size and system software and utilities will improve performance significantly or whether major code redesign is necessary is unknown. The lack of parallel I/O significantly reduces the scaling efficiency shown in Table 2 in production runs; when data are read or written, a single processor does the i/o while the remaining proces-

sors wait. Finally, the need for a two-dimensional domain decomposition to allow finer-grained chunks and therefore better load balancing is obvious as the number of processors employed on a problem increases. The longitudinal data dependencies in SKYHI which must be handled in order to allow such a decomposition have been identified, and it remains to develop a mechanism to deal with them, within the context of a user-friendly model.

Acknowledgments. I would like to thank S. Fan, C. Kerr, J. Mahlman and D. Schwarzkopf for reading the paper and offering valuable comments. Thanks also to the SKYHI user community who have provided me with the feedback necessary to produce a model which is becoming more user-friendly, and to J. Mahlman who recognized the need for a user-friendly SKYHI and who provided me the opportunity to work to that end.

APPENDIX A
TEMPORAL CONTROL OF ARCHIVE FILE FORMS

Six variables are used to control the temporal characteristics of the archive file forms:

(1) the number of times which data is to be written to a file before closing the file and opening another one;
(2) the number of seconds between writes to the file;
(3) the number of time levels that are to be averaged to generate the data that is to be written;
(4) the amount of time to be counted toward the number of seconds between file writes at the beginning of the run;
(5) the time in the run at which the file clock is to start;
(6) the time in the run at which the file clock is to stop.

Variable (1) allows one to write multiple files of a given file form during a job. File names are created automatically following a simple pattern. By making the value of (2) larger than the length of the job, the writing of the particular file form may be turned off. The mechanism to define the time-averaging characteristics are defined by (3); if a snapshot file is wanted, then variable (3) is set to 1. Otherwise, the combination of (2) and (3) determine the frequency of data sampling. Variables (5) and (6) allow one to write a file during a specified period of an integration, and (4) provides a means to write files at the frequency given by (2), but with an offset in time from the start of the job. These six variables are named in a consistent way for each of the file types, differing only by a single letter, which indicates the file type.

APPENDIX B
SUGGESTED PROCEDURE TO USE THE ARCHIVING MODULE

The following process is recommended when using the archiving module:

I) Decide which archive files are to be written during the experiment and define their desired characteristics. The characteristics are found in the namelist and parameter file associated with the given file type.

II) For each file type, choose one of the following options:

A) If no files of this type are desired:

1) Set the preprocessor variable defining the number of file forms of that type to zero.

2) Remove the namelist variables associated with that file type from the namelist.

B) If you desire either a subset or all of the default file forms:

1) Leave the preprocessor file form number variable at the default value.

2) For those file forms that are not being written:

a) Set the seconds between file writes to be larger than the run time of the job.

b) Set variable (3) of Appendix A to be 1, reflecting a snapshot file.

c) Set the spatial index parameters for the file form to all be 1, thus setting the size of the array which will hold the data to be of length 1.

3) For those file forms to be written:

a) Set the namelist and parameter file to contain the desired file characteristics.

b) If the file form containing a subset of the standard file is to be written, modify the subroutine defining the file contents so that it will contain the desired variables .

C) If a new file form is to be added:

1) Set the preprocessor file form number variable to the proper value.

2) Add the code to define the new file form, following the existing code pattern. Code mods must be made to six to ten source files, dependent on the file type.

3) The mods involve duplicating code and changing file form numbers in the variable names to the number of the new form. Additionally the user must define the variable names to be placed in the file, following the provided patterns.

4) Modification to the script will be necessary to save any new files generated and to assign the file characteristics, if desired.

REFERENCES

1. Fels, S.B., J.D. Mahlman, M.D. Schwarzkopf and R.W. Sinclair, 1980: Stratospheric sensitivity to perturbations in ozone and carbon dioxide: radiative and dynamical response. *J. Atmos. Sci., 37,* 2265-2297.

2. Jones, P.W., C.L. Kerr and R.S. Hemler, 1995: Practical considerations in development of a parallel SKYHI general circulation model. *Parallel Computing*, **21**, 1677-1694.

3. Hamilton, K., R.J. Wilson, J.D. Mahlman and L.J. Umscheid, 1995: Climatology of the SKYHI Troposphere-Stratosphere-Mesosphere general circulation model. *J. Atmos. Sci.*, **52**, 5-43.

DESIGN AND PERFORMANCE ANALYSIS OF A MASSIVELY PARALLEL ATMOSPHERIC GENERAL CIRCULATION MODEL

Daniel S. Schaffer * **and Max J. Suárez**

NASA Seasonal to Interannual Prediction Project
NASA Goddard Space Flight Center
Code 971
Greenbelt, MD 20771
dans@janus.gsfc.nasa.gov
+1 301 286-3133
+1 301 286-0240 (FAX)

ABSTRACT

In the 1990's, computer manufacturers are increasingly turning to the development of parallel processor machines to meet the high performance needs of their customers. Simultaneously, atmospheric scientists studying weather and climate phenomena ranging from hurricanes to El Niño to global warming require increasingly fine resolution models. Here, implementation of a parallel atmospheric general circulation model (GCM) which exploits the power of massively parallel machines is described. Using the horizontal data domain decomposition methodology, this FORTRAN 90 model is able to integrate a 0.6° longitude by 0.5° latitude problem at a rate of 19 Gigaflops on 512 processors of a Cray T3E 600; corresponding to 280 seconds of wall-clock time per simulated model day. At this resolution, the model has 64 times as many degrees of freedom and performs 400 times as many floating point operations per simulated day as the model it replaces.

1. INTRODUCTION

The general circulation modeling community constantly demands more computing power to meet its needs. Short to medium range weather forecasters have used increasingly faster machines to run higher resolution models. The improved solutions obtained from higher resolution in numerical weather prediction is well known; Simmons, et al. (1989), among others, document this. Higher resolution is also important to seasonal and interannual variability studies (e.g. Déqué and Piedelievre, 1995 and Lal, et al., 1997). For studies of longer time scale phenomena, completing model runs at any reasonable resolution becomes the challenge. Coupled atmospheric/ocean simulations of El Niño require enormous computational power. Recently, some modelers have turned to ensembles of runs to produce better predictions; a strategy that magnifies resource demands. For the time scales of global climate change, coupled model

*SAIC General Sciences Corporation; Laurel, MD

runs can last hundreds of simulated years (e.g. Manabe and Stouffer, 1994); for studies of the thermohaline circulation, those numbers stretch into the thousands.

To meet these needs, supercomputer manufacturers have offered a variety of solutions. Since the 1980's, parallel vector processors have been the most widely used by the GCM community. However, in the 1990's cache-based massively parallel processor (MPP) machines have become increasingly prominent. These machines present a dual challenge to model designers of writing code that runs efficiently within a single processor yet scales well for hundreds of processors.

A snapshot of the progress of (mostly atmospheric) model designers toward meeting these challenges was presented in a special issue of Parallel Computing in 1995. Drake, et al. (1995) wrote a message passing implementation of the National Center for Atmospheric Research (NCAR) Community Climate Model (CCM2) for the IBM SP2 and Intel Paragon machines. Most notable was the poor single processor performance they attributed to inefficient cache use (a result noted repeatedly in the literature). Jones, et al. (1995) implemented a parallel version of the Geophysical Fluid Dynamical Laboratory (GFDL) Atmospheric General Circulation Model (AGCM) running on the the CM-5 and SGI/Cray C90. Single processing element (PE) performance and scaling were quite good on the C90 but hampered on the CM5 by over-use of memory they attributed to poor algorithmic design. Lou and Farrara (1996) optimized a parallel version of the UCLA AGCM for the Paragon and SGI/Cray T3D/E. The model scales fairly well but their preliminary attempts at cache-based optimizations have yielded modest improvements.

Here we describe the parallel design and performance of an AGCM designed for climate studies. The primary objectives are efficient single PE performance and scalability on MPPs. Section 2 describes the scientific basis of the model. Section 3 explains the high-level model design, the parallelization methodology, and gives highlights of the detailed design. Section 4 analyzes the model performance. Section 5 discusses how the model is currently being used and describes on-going optimization efforts.

2. MODEL DESCRIPTION

The dynamical portion of the GCM is based on a finite-differenced, primitive equations dynamical core (Dycore) (Suarez and Takacs, 1995) that allows arbitrary horizontal and vertical resolution. It is the dynamical core used by Goddard's Data Assimilation Office in the Goddard Earth Observing System (GEOS) GCM and by NASA's Seasonal to Interannual Prediction Project (NSIPP). Its prognostic variables are the two horizontal wind components, the potential temperature, the surface pressure, the water vapor mixing ratio, and an arbitrary number of passive tracers. In the vertical, the discretization scheme closely follows that proposed by Arakawa and Suarez (1983), but applied to a generalized vertical coordinate ($\sigma - p$). In the horizontal, the equations are finite-differenced on a staggered latitude-longitude grid (the C-grid). To avoid linear computational instability due to convergence of meridians

near the poles, a Fourier filter is applied to all tendencies pole-ward of 45 degrees latitude. The model also uses a scale-selective filter (Shapiro, 1970) to damp grid-scale variance that can lead to non-linear computational instability. The model is integrated in time using a leapfrog scheme modified as proposed by Brown and Campana (1978) and by applying a weak time filter (Asselin, 1972).

The solar parameterization (Chou, 1992) models absorption due to O_3, CO_2, water vapor, O_2, clouds, and aerosols, as well as gaseous, cloud, and aerosol scattering. The infrared parameterization (Chou and Suarez, 1994) includes absorption by water vapor, CO_2, O_3, methane, N_2O, CFC-11, CFC-12 and CFC-22 within eight spectral bands. Other parameterizations include the Louis et al. (1982) turbulence and Zhou et al. (1996) gravity wave drag schemes. Penetrative convection originating in the boundary layer is modeled using the Relaxed Arakawa-Schubert (RAS) scheme (Moorthi and Suarez, 1992). The Mosaic land surface model (LSM) (Koster and Suarez, 1992) computes area-averaged energy and water fluxes from the land surface in response to meteorological forcing. A grid square is sub-divided into relatively homogeneous sub-regions ("tiles" of the mosaic), each containing a single vegetation or bare soil type.

3. COMPUTATIONAL DESIGN

We begin by describing the high level structure of the GCM so as to provide context for the results in section 4. The model is divided into self-contained components, each operating on its own space (grid) and time scales. "Coupling" software converts data from one model grid to another in parallel. The couplers serve the same purpose as the NCAR Climate System Model (CSM) flux coupler (Bryan, et al., 1996). The GCM driver that ties together these components can be atmospheric only, ocean only, coupled atmospheric/ocean, etc. Presently, the major components for this AGCM are:

1. Dynamics - Dycore, the Shapiro filter and the model stepping functionality.

2. Slow Physics - The longwave and shortwave radiation calculations.

3. Fast Physics - The remainder of the AGCM; convection, turbulence, land processes, etc.

The parallelization is implemented using a horizontal data domain decomposition. Put simply, each processor operates on a slab of data extending from the surface to the top of the atmosphere. The primary advantage of this decomposition is that the number of horizontal grid points available to divide among the processors is large, allowing utilization of hundreds of PE's. In addition, physics calculations such as longwave, shortwave, etc. become "embarrassingly parallel". Finally, at a practical level, using this scheme means that the original plug compatible physics subroutines can be retained, unmodified, in the parallel implementation.

The processors are laid out in a rectangular array so that each PE has exactly one neighbor on each of four sides. The number of PE's in the X and Y direction (NX and

NY) as well as the number of grid points within each PE (IM and JM) are selectable at run-time. In particular, IM can be different for each of the NX columns of PEs and JM different for each of the NY rows. Ghost (shadow) regions are defined to facilitate local addressing and nearest neighbor communication. When code such as horizontal advection needs to access an array element outside the local processor, a communications call is made to fill in the ghost region. Once the data are in place, the code can process as if it were written for a computer. The communication is bundled over all levels to reduce the impact of latency.

Since the primary objective is implementation on a distributed memory MPP, a message-passing scheme is used for the communication. Generic synchronous point to point send/receive routines provide the backbone for this scheme. Currently they are implemented using calls to either native Cray shared memory software (SHMEM) or message passing interface (MPI) routines. This backbone is packaged into a single "communication primitives" module. Since this is the only code that varies between implementations, porting the model is quite simple.

While most of the communication in the model is nearest neighbor, the polar filter is a significant exception. It is implemented by first transposing the data from an (X,Y) to a (Y,Z) decomposition, then executing local FFTs, then transposing back. This implies that the greater the decomposition in X, the poorer the performance of the polar filter. Conversely, nearest neighbor communication scales as $\frac{1}{\sqrt{PEs}}$ only if the processor layout is close to symmetrical. These conflicting performance considerations guide optimal processor layout choice and represent the most obvious disadvantage of this decomposition strategy.

Currently, no load balancing is implemented. The sources of imbalance are: 1. The shortwave code; radiative transfer calculations need only be performed for sunlit soundings. 2. The land surface code; no computations are needed for ocean points and the uneven distribution of tiles further un-balances the problem. 3. Cumulus convection; fewer computations are needed where convection does not occur. 4. The polar filter; it only operates pole-ward of 45 degrees latitude. The utility of implementing load balancing schemes will be discussed in section 5.

The Dynamics, all upper-level Physics routines and control and communication routines are written in FORTRAN 90. Most of the low-level, plug-compatible, computational routines in the Physics have been left in FORTRAN 77. Array syntax, user-defined types, subroutine overloading, modules, and dynamic memory allocation are used extensively. Use of these features has helped to create reasonably well-structured code and greatly facilitated debugging. Since all memory is dynamically allocated, the model runs at any resolution using any processor layout without recompilation. On the downside, dynamic memory use may hamper future cache-based optimizations.

4. RESULTS AND PERFORMANCE

The model is currently being run on the DEC Alpha workstation, Cray T3E, and Cray J90. To validate the code, results were compared to those from the serial, FORTRAN 77, production version for the same initial and boundary conditions at a resolution of 72x45x22. At this resolution, Dynamics and Fast Physics are run at 9 minute intervals and Slow Physics every 3 hours. After 3 hours, checksums of state variables, budgets and other diagnostic quantities for the old and new code differ at the round-off level; for one or multiple processors.

To assess performance, the floating point operations (FLOPs) are counted for a one processor run on a CRAY J90 using the PERF utility. These numbers are generally more conservative (up to 25%) than the operation counts produced by T3E-native counters. Initialization and finalization times are not counted. No model output is done during the "run" phase for purposes of these benchmarks. Performance is then computed by dividing the FLOP count by the run-time measured by wall-clock timers. The 72x45x22 resolution problem was run on the Cray T3E-600 using 32 bit words for up to 64 PEs. The peak performance is 1.35 Gflop/s, corresponding to 20 seconds run-time per simulated model day. A 64 bit version runs at only 28 seconds per day; largely due to the fact that the code is memory-access bound. In comparison, the original production version running multi-tasked on the Cray J90 (64 bit) simulates one model day in 50 seconds.

To truly exploit the power of the T3E machine, we turn to a high resolution problem; 576x360x22 (0.625° by 0.5° by 22 levels). Preliminary tests show a Dynamics time step of one minute is required to satisfy the Courant-Friedrich-Levy (CFL) condition for linear numerical stability at this resolution. The Fast Physics is run every 10 minutes and Slow Physics at 3 hour intervals. For a 3 hour run, the floating point operations total 686 billion. The 32 bit version requires approximately 1 billion words of memory; translating to a minimum of 64 Cray T3E-600 PEs. The GCM was tested for processor configurations totaling up to 512 PEs. Experimentation showed that for 512 PEs, a processor layout of 16 PEs in longitude, 32 in latitude is optimal. For that case, the performance is 19.6 Gflop/s. This corresponds to 280 seconds of wall-clock time per simulated model day.

The details of the T3E performance are shown in the speedup plots in figure 1. The solid lines in the figure are curve fits of the data to Amdahl's speedup law:

$$S = \frac{1}{Fs + \frac{Fp}{Np}}$$

where S is the speedup, Fs is the serial fraction, Fp is the parallel fraction and Np is the number of processors. For a perfectly load balanced code, the effective single processor performance is an estimate of how fast it would run on 1 PE if that were possible. Notice that, in Dynamics, this number is higher than the per-processor performance because it does not include the degradation due to communication as

Table 1: Floating point operations (in billions), run-time, total performance, and per pe performance for a 3 hour run of the 576x360x22 resolution problem at 512 PEs.

Code	GFLOP	Time (s)	Gf	Mf/PE
Dycore	427.3	17.77	24.1	47.0
Shapiro	74.6	5.56	13.4	26.2
Step	33.6	1.57	21.4	41.8
Longwave	34.3	0.97	35.4	69.1
Shortwave	48.0	2.77	17.3	33.8
Lsm	6.8	0.96	7.1	13.8
Ras	4.6	1.21	3.8	7.4

the problem is scaled to 512 PEs. The floating point operation counts show that Dynamics is responsible for the great majority of the work. This is largely due to its relatively short time steps. The fact that Slow Physics does not scale perfectly is currently under investigation.

Table 1 shows a breakdown of performance of the major GCM components. The dynamical core consumes the most run-time and will need the greatest attention during future optimizations. The poor scaling of the Shapiro filter is expected; it does relatively few floating point operations per communication. That the Step function does not scale perfectly is merely an artifact of the code design. It fills the ghost regions of the state variables; work that could just have easily been done in Dycore.

The LSM and RAS codes are "super-scaling". This commonly observed result occurs because as the number of processors increases, the amount of memory needed per pe decreases and, consequently, the data fit better in cache.

The rated performance of the Cray T3E 600 is 600 Mflop/s. While, in practice, few codes reach 200 Mflop/s per PE, it is clear from table 1 that our per-PE performance is much lower. One reason is poor cache re-use. As a first cut, this code was written to mimic the original serial code which was designed to run efficiently on vector machines. As of yet, no serious cache-based optimizations have been attempted. A second reason is communication costs. Measurements by the T3E Apprentice utility indicate that 25% of the Dycore run-time is communication. Latency is significant. Even with bundled Ghost calls, preliminary measurements indicate that 35% of the nearest neighbor communication time is latency. When the Ghost calls are unbundled, Dycore performance degrades by 20%. A third cause of the poor single-pe performance is load-imbalance as described earlier. Strategies to address these inefficiencies are discussed in the next section.

A Cray J90 SHMEM version of the code for the same resolution performed at 90 Mflop/s on one processor. Since the rated performance of the J90 is 200 Mflop/s, the model is clearly vectorizing quite well. Although a multiple processor J90 version has not been run for this resolution, past experience suggests that it should perform

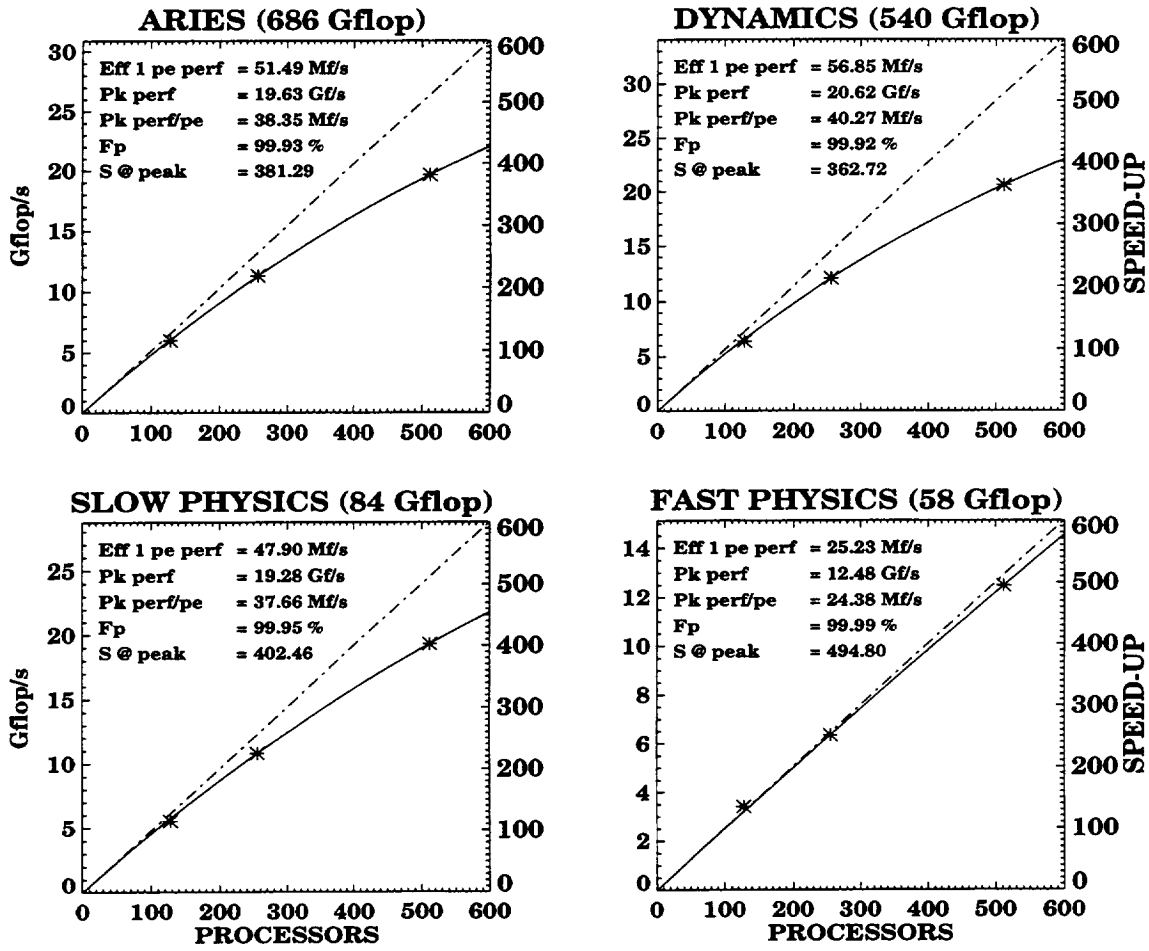


Figure 1: Speedup plots for 3 hour runs of the full GCM and its three major components. The floating point operations in billions are given at the top of each graph. The asterisks represent the speed in Gflop/s for 128, 256 and 512 PEs. The dot-dashed line represents a perfectly linear speedup. The solid curve was obtained by fitting the operations and run-times to Amdahl's speedup law (see text). Fp and S are as given in Amdahl's law. The effective single PE performance is the curve value for Np=1.

at about 1 Gflop/s for 16 PEs. An MPI implementation on the J90 was found to significantly degrade the code's performance; presumably due to the high level of overhead in the MPI software. A T3E MPI version has not been tested.

5. DISCUSSION

As currently written, the code performs well enough to enable production runs at high resolution (0.6° longitude by 0.5° latitude by 22 levels) using 512 processors. In fact, a one year run at this resolution has already been completed. The model can also run efficiently at lower resolutions. For example, a 1.25° longitude by 1° latitude problem running on 128 processors would actually out-perform the high resolution case. The Dynamics, Fast Physics and Slow Physics would run with the same efficiency as in the high resolution case since the amount of work and number of processors have both decreased by a factor of 4. However, for the lower resolution, a Dynamics time step of 2 minutes could be taken, significantly improving the model throughput. The same reasoning applies to a 2.5° longitude by 2° latitude problem running on 32 PEs. For that resolution and lower, an ensemble of runs most effectively utilizes the 512 PE machine. Such ensemble runs are currently underway. Another planned application of the model is to couple it to a parallel version (currently under development) of the Poseidon ocean model (Schopf and Loughe, 1995). A planned 100 year coupled run using a 2.5° longitude by 2° latitude atmospheric resolution running with 32 processors should be quite feasible.

Five major avenues of optimization are under investigation; semi-implicit time differencing, single PE optimization, reduction of software latency in the communication code, load balancing, and parallel/asynchronous I/O. As the results indicate, for the high resolution case, Dynamics is the bottleneck due to the small time step. A semi-implicit time differencing scheme is currently being developed. Successful implementation of this scheme would allow the time step to be raised to perhaps 2-4 minutes for the high resolution problem.

Single PE optimization will largely be achieved by better cache re-use. Preliminary analysis shows that the local storage for one sounding in the longwave code for the high resolution case could fit entirely in cache. Obtaining such a fit should enhance performance. A similar strategy could be applied to the shortwave and Fast Physics codes. Further single PE optimization may require more draconian measures such as re-organizing data structures and writing key components in assembly language. Of course, such modifications would degrade vector performance on parallel vector machines as well as the clarity of the code itself.

As mentioned, communications latency is significant. Much of this latency appears to be due to unnecessary overhead in the "communication primitives" software. Efforts are underway to eliminate this overhead by eliminating communication buffers and superfluous memory access. Elimination of this excess latency should, for example, enable the aforementioned coupled run to scale well beyond 32 processors.

Off-line experimentation suggests that load balancing will improve the performance of the shortwave and LSM calculations. The re-distribution of data is determined ahead of time so the only cost is the actual communication. Some benefit could also be gained from a load-balanced polar filter since at 512 PEs, 60% of the polar filter time is spent doing the actual FFT. For RAS, it is possible no improvement at all will be achieved since a great deal of the run-time would have to spent determining how the data should be re-distributed.

For the long runs currently in progress, relatively little diagnostic output is needed so the cost of I/O is insignificant. It is estimated that even a planned 5-fold increase in model output will not present any great difficulty. Should this turn out not to be the case or if even more extensive output is needed then parallel/asynchronous I/O may be required. Development of parallel I/O software is discussed in Sawyer, et. al. (1998).

In conclusion, a parallel atmospheric general circulation model that successfully exploits the power of MPP's such as the Cray T3E has been developed. The model is being used for high resolution runs as well as ensembles of low resolution cases. On-going efforts to improve single processor performance, reduce communication overhead, and mitigate load imbalancing will enable even more effective use of these powerful machines.

ACKNOWLDEGEMENTS

This project is supported by the Global Modeling and Analysis Program in NASA's Office of Mission To Planet Earth under RTOP No. 622-24-47. Access to the Cray T3E-600 was provided by the Earth and Space Sciences (ESS) component of the NASA High Performance Computing and Communications (HPCC) Program. We would like to acknowledge Jim Abeles of SGI/Cray for the help he has provided over the years in design and optimization. Thanks also go to Tom Head of Carnegie Mellon for early work on the project and to Paul Schopf of George Mason University for his ideas on the GCM design.

REFERENCES

Arakawa, A. and Suárez, M.J., 1983: Vertical differencing of the primitive equations in sigma coordinates. *Mon. Wea. Rev.*, **111**, 34-45.

Asselin, R., 1972: Frequency filter for time integrations. *Mon. Wea. Rev.*, **100**, 487-490.

Brown, J.A. and Campana, K., 1978: An economical time-differencing system for numerical weather prediction., *Mon. Wea. Rev.*, **106**, 1125-1136.

Bryan, F.O., Kauffman, B.G., Large, W.G., Gent, P.R., 1996: The NCAR CSM Flux Coupler. *NCAR Technical Note* (NCAR/TN-424+STR, May 1996).

Chou, M.-D., 1992: A solar radiation model for use in climate studies. *J. Atmos. Sci.*, **49**, 762-772.

Chou, M.-D. and Suárez, M.J., 1994: An efficient thermal infrared radiation parameterization for use in general circulation models. *NASA Technical Memorandum*, **3**, 104606, 84pp.

Déqué, M. and Piedelievre, J. Ph., 1995: High Resolution climate simulation over Europe. *Climate Dynamics*, **11**, 321-339.

Drake, J., Foster, I., Michalakes, J., Toonen, B., and Worley, P., 1995: Design and performance of a scalable parallel community climate model. *Parallel Computing*, **21**, 1571-1591.

Jones, P.W., Kerr, C.L., Hemler, R.S., 1995: Practical considerations in development of a parallel SKYHI general circulation model. *Parallel Computing*, **21**, 1677-1694.

Koster, R.D. and Suárez, M.J., 1992: Modeling the land surface boundary in climate models as a composite of independent vegetation stands. *J. Geophy. Res.*, **97**, 2697-2715.

Lal, M., Cubasch, U., Perlwitz, J., and Waszkewitz, J., 1997: Simulation of the Indian Monsoon Climatology in ECHAM3 Climate Model: Sensitivity to Horizontal Resolution. *Intl. J. Climat.*, **17**, 847-858.

Lou, J. and Farrara, J., 1996: Performance Analysis and Optimization on the UCLA Parallel Atmospheric General Circulation Model Code. In *Proceedings Supercomputing '96*, Pittsburgh, PA, USA, ACM-IEEE.

Louis, J., Tiedke, M., and Geleyn, J., 1982: A short history of the PBL parameterization at ECMWF. In *ECMWF workshop on Planetary Boundary Layer Parameterization*, Reading, pp. 59-80.

Manabe, S. and Stouffer, R.J., 1994: Multiple-Century Response of a Coupled Ocean-Atmosphere Model to Increase of Atmospheric Carbon Dioxide. *J. Climate*, **7**, 5-23.

Moorthi, S. and Suárez, M.J., 1992: Relaxed Arakawa-Schubert: A parameterization of moist convection for general circulation models. *Mon. Wea. Rev.*, **120**, 978-1002.

Sawyer, W., Lucchesi, R., Lyster, P.M., Takacs, L.L., Larson, J., Molod, A., Nebuda, S., and Pabon-Ortiz, C. 1998: Parallelization aspects of an atmospheric general circulation model for data assimilation. In *Proceedings High Performance Computing '98*, Boston, MA, USA.

Schopf, P. and Loughe, A., 1995: A reduced-gravity isopycnic ocean model - hindcasts

of El Niño. *Mon. Wea. Rev.*, **123**, 2839-2863.

Shapiro, R., 1970: Smoothing, filtering and boundary effects. *Rev. Geophys. Space Phys.*, **8**, 359-387.

Simmons, A.J., Burridge, D.M., Jarraud, M., Girard, C., and Wergen, W., 1989: The ECMWF Medium-Range Prediction Models Development of the Numerical Formulations and the Impact of Increased Resolution. *Meteorol. Atmos. Phys.*, **40**, 28-60.

Suárez, M. J. and Takacs, L.L., 1995: Documentation of the Aries/GEOS dynamical core Version 2, NASA Technical Memorandum, **5**, 104606, 58pp.

Zhou, J., Sud, Y.C., and Lau, K.-M., 1996: Impact of orographically induced gravity-wave drag in the GLA GCM. *Quart. J. Roy. Meteor. Soc.*, **122**, 903-927.

Requirements and Problems in Parallel Model Development at DWD

Ulrich Schättler, Günther Doms
Deutscher Wetterdienst, Postfach 100465, 63004 Offenbach, Germany
uschaettler@dwd.d400.de
phone: +49 69 8062-2739; fax: +49 69 8236-1493

Abstract

Nearly 30 years after introducing the first computer model for weather forecasting, the *Deutscher Wetterdienst* (DWD) develops the 4th generation of its numerical weather prediction (NWP) system. It consists of a global grid point model (GME) based on a triangular grid and a non-hydrostatic *Lokal Modell* (LM). The operational demand for running this new system is immense and can only be met by parallel computers.

Regarding former NWP models, several new problems had to be taken into account during the design phase of the system. Most important are the portability (including efficieny of the programs on several computer architectures) and an easy code maintainability. Also the organization and administration of the work done by developers from different teams and institutions is more complex than it used to be.

This paper describes the modular approach used for the design of the LM and discusses the effects on the development. Some results of investigations from GMD (German National Research Center for Information Technology) and the software engineering company Pallas are presented on how the LM will perform on different computer architectures and how new hardware will influence the programming style used.

1 Introduction

In 1996 DWD started to develop the 4th generation of its NWP system. The current 3rd generation operational system consists of a spectral *Global Modell* (GM), a regional grid point model for the synoptic and meso-α scale covering the Northern Atlantic and Europe (the *Europa-Modell* EM) and a high resolution meso-β scale *Deutschland-Modell* (DM). EM and DM are running the same code but with different domain sizes and resolutions.

In the new system, GM and EM are combined to a global grid point model GME with physical packages based on the EM/DM. It should produce global forecasts for up to 7 days at least in the quality of the EM. The hydrostatic DM will be replaced by a non-hydrostatic *Lokal Modell* (LM), which will be used for numerical weather prediction on the meso-β and on the meso-γ scale as well as for the evaluation of local climate and for various scientific applications covering a wide range of spatial scales (down to grid spacings of about 100 m). The weather forecasts include clouds, fog, precipitation, local wind systems and also severe weather events. The whole system will be used as simulation and research tool for applications such as parameterizations, data assimilation and climate investigations. For the development of both models collaborations have been started with several national and international research institutes and universities.

The initial resolutions of the models for NWP ($\sim$ 55 km horizontal for GME with 31 levels and $\sim$ 8 km for LM with 35 levels) will be increased in the next years (to $\sim$ 25 km for GME with 40 levels and $\sim$ 2-3 km for LM with 50 levels) demanding a computational power of about 300×10^{12} floating point operations for a 24 hour forecast for each model. To meet these requirements, GME and LM have been parallelized and implemented for distributed memory parallel computers using Standard Fortran 90 and the Message Passing Interface (MPI) as parallel library. But they can still be executed on conventional scalar and vector computers where MPI is not available.

With such a computer power necessary, the efficiency of the models is extremly dependent on the underlying hardware. Changes to computer and processor architectures in the past have forced model developers to a total restructuring and recoding of their codes. With the development speed of computers in mind it can be foreseen that the frequency of such updates will be increased in the future. On the other hand it is not clear today which computer or processor architecture will be the most promising or affordable one in about 3-5 years. To the well known requirements of code maintainability and efficiency on one computer system now also comes the portability to and the efficiency on a wide range of different computer systems and architectures. At the same time the program design should also allow for easy code modifications to react not only on changes in computer hardware but also on new scientific developments.

Another problem of the model development is that only few scientists involved do have experience in parallel programming or in the new features of Fortran 90. Therefore, a good strategy has to be implemented that enables also programmers with only few knowledge to work on the code.

This paper reports on the development progress reached so far at DWD. Section 2 gives the basic features and parallelization strategies of both models. The modular approach used for the design of the LM is described in Section 3. The effect of the modularity on the development work today and in the future is discussed. Some future problems regarding computer architecture and programming style are presented in Section 4.

2 Description of the Models

Detailed scientific documentations are available for both models [1, 2]. Therefore, only some basic features will be given here. A more comprehensive summary can be found in [3].

2.1 The Nonhydrostatic Regional Model LM

Equations, algorithms and grid structure
The LM is based on the primitive hydro-thermodynamical equations describing compressible nonhydrostatic flow in a moist atmosphere without any scale approximations. A basic state is subtracted from the equations of motion to reduce numerical errors associated with the calculation of the pressure gradient force in case of sloping coordinate surfaces. The continuity equation is replaced by an equation for the perturbation pressure, which becomes a prognostic variable besides the three velocity components, temperature, water vapour and cloud water. The set of model equations is formulated in rotated geographical coordinates and a generalized terrain following vertical coordinate.

Spatial discretization is by standard second order finite difference schemes on a C-/Lorenz-grid. The time integration is performed with the leapfrog-method using Klemp's and Wilhelmson's [4] time splitting technique including extensions proposed by Skamarock and Klemp [5] to solve for the sound and gravity wave terms. The idea of the time splitting is to treat the fast terms describing sound and gravity wave propagation with small time steps Δt_s while doing a large step Δt for the slow terms (advection, physics). Because only a subset of terms in the model equations is integrated with a small Δt_s whereas the computational expensive slow terms have to be updated less frequently, the algorithm becomes numerical efficient.

The physics package of LM has been adapted from the operational hydrostatic DM and thus only applies on the meso-β but not on smaller scales. Work on new parameterization schemes to upgrade the physics for model applications on smaller scales is in progress.

Parallelization
The parallelization strategy for the LM is the 2D domain or data decomposition (grid partitioning) which is well suited for grid point models using finite differences. This strategy also is used and described by several other authors [6, 7, 8, 9]. Each processor gets an appropriate part of the data to solve the model equations on its own subdomain. These subdomains are arranged in a two-dimensional array of rectangular tiles. The local data structure of every processor contains additional rows and columns to store the values of grid points belonging to neighboring processors (see Figure 1). During the integration step each processor updates the values of its local subdomain; grid points on the edges are exchanged using explicit message passing.

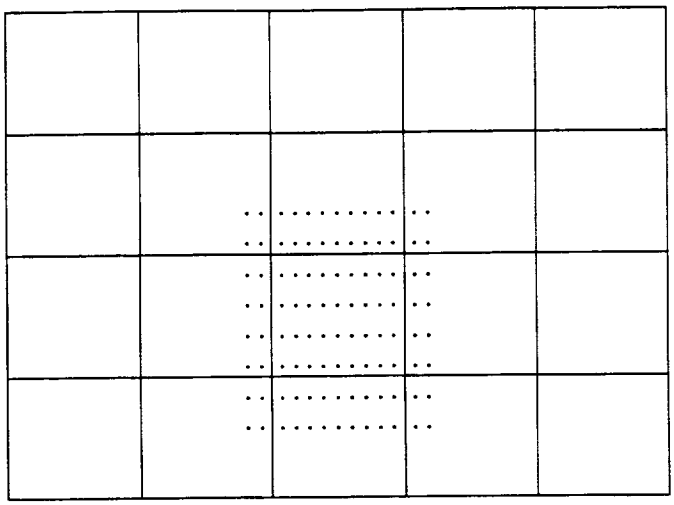


Figure 1: 2D domain decomposition with local data structure

The splitting method used in the LM is implemented with an implicit Crank-Nicolson method in the vertical and with an explicit forward-backward scheme in the horizontal. Therefore, only a nearest-neighbor exchange, i.e. local communication, is necessary for this model.

2.2 The new Global Model GME

Equations and algorithms
The system of equations solved in the GME is based on the hydrostatic primitive equations. These equations are integrated using a semi-implicit algorithm. The explicit part is solved with a three time-level semi-Lagrangian scheme while the semi-implicit corrections are computed by solving an elliptic PDE, namely a Helmholtz-equation.

Grid Generation
For the horizontal discretization of the equations a triangular grid based on the icosahedron is introduced. It was first described by Sadourny et.al. [10] and Williamson [11]. The approach outlined here is based on the work of Baumgardner [12]. The same grid today also is used by Loft [13].

To construct the grid, the sphere is divided into 20 spherical triangles of equal size by placing a plane icosahedron into it. The 12 vertices of the icosahedron touch the sphere, one vertex coincides with the north pole and the opposite one with the south pole. The spherical triangles are defined by the great circles connecting two vertices respectively. Each of the 12 vertices then is surrounded by 5 spherical triangles. Two adjacent triangles are combined to form a "diamond", i.e. a logically square block.

For further grid generation, the sides of the 20 main triangles are subdivided iter-

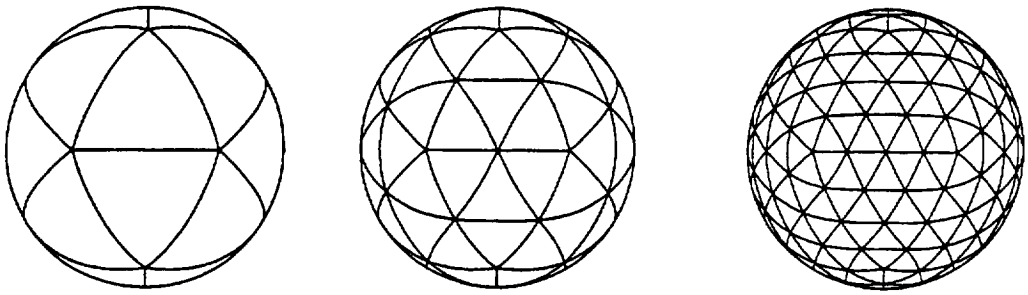


Figure 2: Grids derived from the icosahedron

atively into ni equal parts to form subtriangles. Each point in a main triangle is surrounded by six triangles and accordingly is in the center of a hexagon. However, the points which form the vertices of the icosahedron are surrounded by only five triangles and therefore are the centers of pentagons. Some resulting grids are illustrated in Figure 2.

The derivation of the necessary numerical operators (e.g. for the gradient, the divergence or the Laplacian) for this triangular grid as well as a more detailed explanation of the grid generation can be found in the documentation of the GME [2].

Parallelization
The diamonds can be looked upon as logical square blocks and therefore can be implemented with normal data structures. In the sequential program a global twodimensional field is stored as a threedimensional array. The third dimension represents the 10 different diamonds covering the earth.

The parallelization strategy is by data decomposition again. But while this is straightforward for a regional model a more sophisticated strategy has to be used here. A practical way for the parallelization is based on the viewpoint that every diamond can be regarded as a regional model and is related to an idea of John Baumgardner. Every diamond can be partitioned in the same way like the domain of a regional model. Since all diamonds are of equal size, their decomposition is identical. If the processors are arranged in a two dimensional grid corresponding to the decomposition of the diamonds, every processor gets a part of each diamond. This kind of decomposition is shown in Figure 3 for 4 processors.

Other decompositions of this triangular grid that minimize the amount of data to be transferred have been investigated by GMD [14, 15]. But within the decomposition described above the 10 parts of each diamond that a processor gets are distributed regularly over the earth. From a statistical point of view there is a chance to get a rather balanced load distribution, regarding the computations in the physical packages (day-night radiation, land-water distribution).

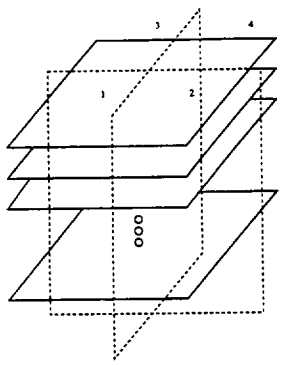


Figure 3: Logical view of the decomposition for the GME

3 Code Design for the LM

Modular design
Modularity is a basic attribute of NWP models, but in programming languages as Fortran 77 it was difficult to express in the program design. Fortran 90 supports a modular development approach by grouping together variable declarations and subprograms into `MODULE`s.

The LM uses `MODULE`s in three different ways:

- The data modules form the data pool of the model (meteorological as well as organizational variables). With the Fortran 90 `USE`-statement these data are available for other modules. The data modules replace the `COMMON`-blocks used in Fortran 77.

- The second group of modules provide utility routines that handle small tasks which need not be model specific. Examples are the time measurement, the determination of the actual date and time or the computation of meteorological variables derived from the prognostic variables. All routines necessary for parallel programming (i.e. routines containing calls to the message passing library) are also put into utility modules.

- All routines belonging to a model specific task (or package) are combined in a source module. "Package" is a term defined by Kalnay [16] regarding the physical packages, i.e. the parameterization of the atmospheric subgrid-scale physical processes such as radiation or convection. More general, also other parts of the model (dynamics, input and output of data) can be viewed as packages. By using the data and utility modules the source modules belonging to these packages can be written in a way that they are independent from each other.

Every module has to list the data and the routines used from other modules. These lists define clearly the interface of the module. Figure 4 shows the modules used in the LM and their dependencies. The top level of the model is the main program `lmorg`. It manages all tasks of the forecast by using the source modules.

The clear modular formulation facilitates concurrent work on different (source) modules. For the development of the LM this is very important, because most physical parameterizations have to be adapted to very high resolutions in the next months. The work on the different schemes can be done without conflicting others. At the same time also different numerical schemes for the dynamics are investigated and tested.

Portability
One of the main goals for the source code development of the LM is portability. First of all this means that the same code has to run on every computer platform without having to change the source itself. This is obtained by using only standard Fortran 90 and MPI. MPI is adopted as a standard by nearly all computer vendors and efficient implementations are available for their parallel machines. For sequential platforms having no MPI implementation, dummy interfaces for the MPI routines are provided for the LM.

A second aspect of portability is that the program should also be efficient on different machines. The efficiency of the LM on vector processors is very good, because the code is written in the same way as former highly vectorized models (the most inner loop is horizontal east-west direction). The coding style used for vectorization has some limitations for cache based scalar RISC processors. Many compilers on the other hand have optimization features (such as loop unrolling or splitting, etc.) that can produce rather efficient code for RISC processors. The optimizations performed on the LM code are not hardware specific, but in a way that every processor architecture will benefit (avoid duplicate computations in different routines by providing more memory; avoid divisions, etc.)

Communication is not a very critical issue for the LM. During a time step only local communications with the neighbors are necessary. To compute some mean values for monitoring the forecast, a global reduce is done from time to time. On all machines and with all domain sizes tested so far, the communication time was below 3 % of the total time.

Up to now the LM has been tested on Cray PVP machines, Cray T3E, SGI Origin 2000, IBM SP2, Fujitsu VPP700, NEC SX-4 and several workstations.

Parallel Programming
As mentioned above, a portable parallel version of the LM is already available. Research and development is going on in the parameterizations, the dynamics, the assimilation scheme and related areas. The problem now faced is that most of the programmers involved in this work do not have much experience in parallel program-

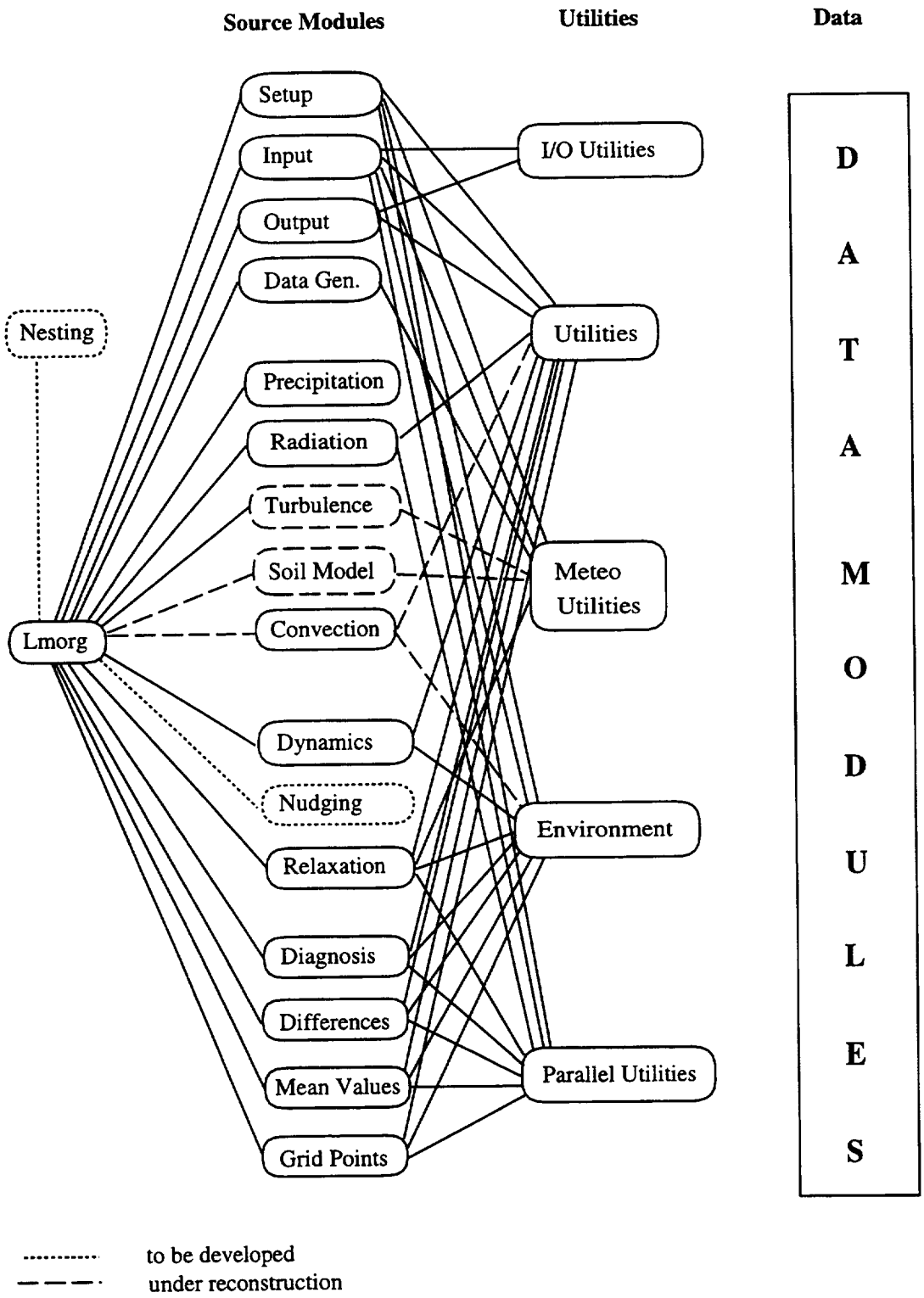


Figure 4: Modular structure of the LM

ming. Therefore, a strategy has been thought of, to enable them to work on the parallel LM and in a parallel computing environment.

The basic idea is that the computations in a subdomain are organized in the same way as the ones in the total domain, if working on a shared memory computer. The total domain has a specified number of boundary lines (= `nboundlines`) at each side, on which values are provided by the surrounding model. The forecast is computed only in the interior of the total domain. The same holds for every subdomain, with the exception that the values on the boundary lines (also `nboundlines` at each side) are provided by the neighboring processors via message passing.

Three different kinds of calculations have to be considered for the programming:

- Loop organization: The horizontal size of a subdomain is (`1...ie,1...je`). Start- and end-indices are provided, if values have to be calculated only in the interior part (`istart...iend,jstart...jend`). If values have to be calculated also on the boundaries, the loops range from `1...ie` and `1...je`, respectively. These values are set at the beginning of the program, according to the number of processors and the decomposition. Therefore, for most loop calculations there is no difference between the sequential and the parallel program.
- Grid point calculations: To perform computations on certain grid points, routines are provided to determine the local indices and the number of the subdomain in which a grid point is located from the global indices of the total domain and vice versa.
- Elemental parallel operations: Routines for special operations needing message passing are included in the utility modules. These are tools for computing e.g. mean or extreme values of the total domain as well as distributing values to or collecting them from the nodes.

The features described above allow programmers to work on special modules of the LM in a parallel environment without having much knowledge in parallelization. They are able to get the code running, but an experienced programmer might have to optimize the modules later on.

4 Future Development Problems

The LM will be the main forecast tool of DWD in the next decade. In 2001 it should run with a grid size of $800 \times 800 \times 50$ points and a time step of $\Delta t = 10s$. A 24 hour forecast has to be completed during 30 minutes of wallclock time. For that purpose, the computing power at DWD is increased in the next years. The current SGI/Cray T3E with 376 application processors will be replaced by a system with

Table 1: Predicted timings for different processor speeds

Processor	runtime (h)	dynamics %	physics %	MPI %	Efficiency
T3E600	4.78	64.83	33.64	1.53	0.86
T3E900	3.20	64.63	33.53	1.84	0.86
T3E4000	0.72	61.20	31.75	7.05	0.81

1024 processors in 1999 and later on by a successor system, the architecture of which is not clear today.

A current trend on the hardware sector is the clustering of SMP (symmetric multi processing) systems. DWD now is concerned about the performance of the LM on machines with ≥ 1000 processors and about such SMP clusters. Also it is very important to know whether the programming style has to be changed to fully exploit the two different connection systems (inter- and intranode communication) of clusters. Similar problems have been studied e.g. by [17] and [18].

These questions have been investigated on behalf of DWD by GMD and the software engineering company Pallas [19, 20]. They constructed a run time model for the LM and predicted the performance on several partly non-existing computer architectures.

Table 1 shows predicted runtimes of the LM in the size described above for a 24 hour forecast on a 1024 processor T3E with different processor power (T3E600 with 600 MFlop/s peak performance, T3E900 and a fictitious T3E4000). Given are the runtime in hours and the percentages for the computations (in the dynamics and in the physics) and for the communications together with the parallel efficiency. The same interconnection network has been assumed for all processor types, therefore the percentage of the communication is higher for faster processors resulting in a decreased efficiency. Table 1 shows that the processor speed has to be about 7 times faster than that of the T3E600 to compute a 24 hour forecast in half an hour.

One way to reach the desired speed within one processing element is the utilization of SMP nodes. As programming models for SMP clusters there are two major alternatives:

- Only message passing on all processors:
  This will be efficient, if the MPI implementation can fully exploit the speed of the shared-memory communication within one SMP-node. For the LM this model has the advantage, that no changes are necessary.
- Message passing on the cluster level and shared-memory programming within one node:
  The shared-memory programming could be done with automatic parallelization,

which most compilers provide on loop level. The code could also be taken as it is today, but normally this is not very efficient. By using compiler directives, the efficiency will be better, but major changes to the code are necessary then. Another problem of this approach is the portability, but OpenMP could be a new standard for the parallelization with directives.

Again, the modular design of the LM would facilitate the adaption to SMP-clusters using the shared-memory model, because an incremental parallelization is possible, starting with the most computing intensive modules.

References

[1] G. Doms and U. Schättler, The Nonhydrostatic Limited-Area Model LM (Lokal Modell) of DWD – Part I: Scientific Documentation, *Technical Report*, DWD, March 1997.

[2] D. Majewski, Documentation of the New Global Model GME, Deutscher Wetterdienst, 1996.

[3] U. Schättler and E. Krenzien, Model Development for Parallel Computers at DWD, *Making its Mark – Proceedings of the Seventh ECMWF Workshop on the Use of Parallel Processors in Meteorology,* G.-R. Hoffmann and N. Kreitz, eds. (World Scientific 1997) 83-100.

[4] J.B. Klemp and R.B. Wilhelmson, The Simulation of Three-dimensional Convective Storm Dynamics, *Journal of the Atmospheric Sciences*, 35 (1978) 1070-1096.

[5] W. Skamarock and J.B. Klemp, The Stability of Time-splitting Methods for the Hydrostatic and Nonhydrostatic Elastic Systems. *Monthly Weather Review* 120 (1992) 2109-2127.

[6] A. Dickinson, P. Burton, J. Parker and R. Baxter, Implementation and Initial Results from a Parallel Version of the Meteorological Office Atmosphere Prediction Model, *Coming of Age – Proceedings of the Sixth ECMWF Workshop on the Use of Parallel Processors in Meteorology,* G.-R. Hoffmann and N. Kreitz, eds. (World Scientific 1995) 177-194.

[7] I. Foster and J. Michalakes, MPMM: A Massively Parallel Mesoscale Model, *Parallel Supercomputing in Atmospheric Science – Proceedings of the Fifth ECMWF Workshop on the Use of Parallel Processors in Meteorology*, eds. G.-R. Hoffmann and T. Kauranne, (World Scientific 1993) 354-363.

[8] T. Kauranne, J. Oinonen, S. Saarinen, O. Serimaa and J. Hietaniemi, The Operational HIRLAM 2 Model on Parallel Computers, *Coming of Age – Proceedings of the Sixth ECMWF Workshop on the Use of Parallel Processors in Meteorology,* G.-R. Hoffmann and N. Kreitz, eds. (World Scientific 1995) 63-74.

[9] U. Schättler and E. Krenzien, The Parallel "Deutschland-Modell" — A Message-Passing Version for Distributed Memory Computers, *Parallel Computing*, 23 (1997) 2215-2226.

[10] R. Sadourny, A. Arakawa and Y. Mintz, Integration of the Nondivergent Barotropic Vorticity Equation with an Icosahedral-Hexagonal Grid for the Sphere, *Monthly Weather Review* 96 (1968) 351-356.

[11] D. Williamson, Numerical Integration of Fluid Flow over Triangular Grids, *Monthly Weather Review*, 97 (1969) 885-895.

[12] J. Baumgardner, A Three-Dimensional Finite Element-Model for Mantle Convection, Ph. D. thesis, The University of California at Los Angeles, 1983.

[13] Richard D. Loft, A Modular 3-D Dynamical Core Testbed, *Making its Mark - Proceedings of the Seventh ECMWF Workshop on the Use of Parallel Processors in Meteorology,* G.-R. Hoffmann and N. Kreitz, eds. (World Scientific 1997) 270-283.

[14] O. Bröker, K. Cassirer, R. Hess, C. Jablonowski, W. Joppich and S. Pott, Forschungs- und Entwicklungsarbeiten im Rahmen des neuen Global-Modells (GME) des DWD, Gesellschaft für Mathematik und Datenverarbeitung, Internes Arbeitspapier, 28. November 1996.

[15] O. Bröker, Laufzeitvorhersagen für parallele Versionen des globalen Wettermodells GME, Diplomarbeit, Rheinische Friedrich-Wilhelms-Universität Bonn, März 1998.

[16] E. Kalnay et.al., Rules for Interchange of Physical Parameterizations, *Bull. A.M.S.*, 70 (1989) 620-622.

[17] Chris N. Hill and Andrew Shaw, Transitioning from MPP to SMP: Experiences with a Navier-Stokes solver, *Making its Mark - Proceedings of the Seventh ECMWF Workshop on the Use of Parallel Processors in Meteorology,* G.-R. Hoffmann and N. Kreitz, eds. (World Scientific 1997) 250-269.

[18] Aaron C. Sawdey, Matthew T. O'Keefe and Wesley B. Jones, A General Programming Model for Developing Scalable Ocean Circulation Applications, *Making its Mark - Proceedings of the Seventh ECMWF Workshop on the Use of Parallel Processors in Meteorology,* G.-R. Hoffmann and N. Kreitz, eds. (World Scientific 1997) 209-225.

[19] Untersuchung und Modellierung des Lokalen Modells (LM) für Cluster paralleler Systeme mit gemeinsamem Speicher, GMD - Forschungszentrum Informationstechnik GmbH, Institut für Algorithmen und Wissenschaftliches Rechnen (SCAI), Schloß Birlinghoven, 53754 Sankt Augustin.

[20] Der DWD LM-Code für SMP-Cluster: Leistungsschätzung und Untersuchungen zur Programmierung, PALLAS GmbH, Hermülheimerstraße 10, 50321 Brühl.

Experiences with parallelisation of the Unified Model at the UK Met. Office

Author: Rick Rawlins
Meteorological Office
London Road
Bracknell RG12 2SZ
United Kingdom
email: frawlins@meto.gov.uk
phone: 44 01344-856482
fax: 44 01344-854026

1. Introduction

1.1 Background

The Unified Model (UM) is the suite of software developed and used at the Met. Office for atmosphere and ocean numerical modelling. The UM is the central component both of operational numerical weather forecasts, and of all climate prediction research studies carried out at the Met. Office (within the Hadley Centre). A range of temporal and spatial scales, including global and regional domains, is supported in the formulation of the UM and allows it to be used in a number of different model configurations, for a variety of operational and research activities. Following the introduction of the UM into operational service in 1991, both its formulation and capabilities have been significantly enhanced. Parallelisation of the model to take advantage of distributed memory platforms is a continuation of this process.

The code structure of the UM was developed in Fortran77 with standard extensions, including allocation of dynamic memory. I/O operations are coded in C as an aid to portability across platforms. Original developments and operational running began on an 8 CPU Cray-YMP, subsequently to be replaced by a 16 CPU Cray-C90. For these shared memory architectures, parallelisation was achieved by autotasking through pre-processor directives, with little involvement by programmers. Most programming effort for optimisation went into maximizing vector lengths and ensuring that DO loops vectorized. In order to take advantage of a massively parallel computer (MPP) with distributed memory, it was necessary to make substantial changes to the code. Hence a migration strategy was planned for a project that would take several years, starting from a team of 3 and building to include over 50 staff at the height of development activities.

1.2 Hardware

A Cray-T3E was installed at the Met. Office in September 1996, with successive hardware upgrades leading to the present configuration of 870 processors. Model runs of the UM are launched from a graphical user interface serving a local network of Hewlett-Packard workstations. Operational and climate production output data are transferred to an IBM 9672 R73 general purpose data server for subsequent post-processing and archiving functions.

The Cray-T3E is used for a variety of jobs, sharing between operational and climate users in the Met.Office. To make most efficient use of the MPP resource, the operational global model is run with

144 processors, and each standard climate production model is run with 72 processors, although higher numbers of processors have been used in a 'catch-up' mode.

All functions of the Cray-C90 were progressively mirrored on the Cray-T3E during 1997. Production of operational forecasts was switched over in January 1998 and the Cray-C90 dismantled and removed in February.

1.3 Requirements for parallelisation

The Met. Office has a rolling programme to improve operational scores and products whilst preserving the timeliness of its services. This can only be achieved by a combination of higher resolution models, enhanced physical parametrization and more accurate numerical solvers, all of which require more computing power. In particular, a significant benefit had been evaluated from improving the resolution of the operational global model [from 19 to 30 vertical levels, from 90 km to 60 km in the horizontal]. The next generation of coupled climate model required a higher resolution ocean model component, increasing the number of horizontal points by x6, with a target of completing 4 model years per day on 72 processors. Overall this leads to a need for an increase in processing power by a factor of an order of magnitude.

It was essential to preserve existing code structure where possible to allow parametrization enhancements already under development to be integrated into the code simply, to avoid the generation of new errors and the need for repeats of costly development tests. For both operational and climate models, it was vital to validate results obtained on the Cray-T3E in comparison with those on the Cray-C90.

Although code development was performed on the Cray-T3E, it was important to retain the portable capability already established for the UM. This now needed to include the option of running in either MPP or shared memory modes.

2. Parallelisation in the Unified Model

2.1 Basics

Parallelisation is achieved by regular domain decomposition of the grid point array with message passing between domains accessed through a generalised set of library interface routines (GCOM), initially developed at SINTEF. Atmosphere and ocean model configurations of the UM adopt different horizontal domain decomposition strategies: into 2-D arrays and 1-D rows respectively. Message passing is minimised by the introduction of halos around each horizontal domain. Segments of global rows and columns are allocated for each processor simply by dividing the global domain geometrically, starting from the NW corner: each processor will command a similar, but not necessarily identical number of points.

Early investigations with parallel code in the UM used PVM as the underlying message passing method. In fact the very first runs of a parallel UM were achieved on a distributed network of Hewlett-Packard workstations. Later development switched to Cray-specific message passing as performance was ramped up to meet production targets. It was found that the overhead of the GCOM interface layer was significant for some classes of data transfer and explicit alternatives were written directly with Cray-specific message passing, but retaining the option of GCOM for portability. In particular a routine

for swapping halo information between domains is frequently called and generalised, but needed to be re-written for maximum efficiency.

Although the general strategy of domain decomposition was maintained through most of the code, specific problems with global processes were resolved by solutions which involved a gathering and re-scattering of data domains. At the top level of code a COMMON block is populated with information depending on the processor configuration, ie the number of processors in x and y directions. The position of each processor is identified with logical variables such as "atbase" and "attop", and key size information for both global and local arrays is also held.

The compiler changed from Fortran77 on the Cray-C90 to Fortran90 on the Cray-T3E, but none of the extra features of Fortran90 were adopted beyond the Fortran77 extensions already normally available. This was necessary to preserve a portable capability and to reduce the extent of code conversion. Hence DO loop limits remain explicit. It was found to be advantageous to harmonise the use of DO loop limits by adopting a common set of variables such as "START_POINT_NO_HALO" which could be used generally at lower levels of the code once initialised at an intermediate control interface.

Owing to the need to introduce developmental changes in a large and evolving suite of software, code for MPP was introduced under a compile time switch. This allowed a preliminary set of changes to be included in the main body of code for bench-marking and other investigations before every functional area affected by parallelisation needed to be completed.

2.2 Atmosphere model

Primitive equations for model dynamics are solved on a regular latitude-longitude horizontal grid with hybrid vertical coordinates, using a conservative split explicit scheme with a Heun timestep for advection. Second or fourth order accuracy is supported, the latter requiring a double width halo to cater for the extra horizontal data dependency. Routines for gathering, scattering and swapping halo information were written as general purpose routines and it is possible to make these changes simply in the code. However, to minimize the extra costs of fourth order accuracy it was also found necessary to tailor code locally to account for the extra halo width explicitly.

Filtering of model increments is performed for pole-ward rows, with an equator-ward extent dependent on maximum wind speeds. This process was parallelised across global rows and vertical levels, re-distributing the data globally.

Negative humidity amounts can be generated by the advection scheme and in the original non-MPP code these are zeroed while retaining global budgets by sharing accumulated deficits over all points in the layer. This is an expensive method for MPP code and an alternative scheme was developed in which the deficit was distributed over an array of near neighbouring points, which would normally confine calculations to the local or adjacent processor. Model physics parametrizations had few horizontal dependencies and so domain decomposition was relatively straightforward. Exceptions were parts of the boundary layer code and convective momentum transport, which required interpolation between wind and temperature staggering on the Arakawa 'B' grid. Most work concentrated on optimising single processor performance for the compiler. The method of gathering over land points for physics calculations of land processes was maintained for MPP coding, the only change needed being to identify any all-ocean processors with no land points as special cases.

The standard operational global model runs with 144 processors (16 N-S by 9 W-E) which is a compromise between reducing communications between processors and improving the load balancing of the physics. The latter depends on the geographical distribution of sunlit points for radiation calculations and of thermodynamically unstable points for convection calculations.

Assimilation of observations was parallelised by first distributing observations across all processors for preliminary processing, but with load balancing of horizontal interpolation of model increments achieved over levels.

2.3 Ocean model

The ocean model domain decomposition is 1-D, ie divided into latitudinal rows. This limits the maximum number of processors available for a model run to the number of rows: for the ocean submodel within the current climate production model (HADCM3) this is 144.

Filtering was found to lead to a strong loading imbalance and it was necessary to redistribute work, with each vertical level of a filtered row being assigned to successive processors. Timings were originally dominated by the Laplacian solver for streamfunctions, due to a high cost of communications, with halo swapping and global sums being computed for each iteration of the solver. This was modified so that each island summation was performed concurrently on different processors and requires that the number of processors is at least as large as the number of islands for successful load balancing.

2.4 Atmosphere-Ocean coupled model

Atmosphere and ocean submodels run asynchronously within a single Fortran program, with a parallel atmosphere submodel followed by a parallel ocean submodel, each using the same number of processors. HADCM3 has different horizontal dimensions for atmosphere (96 columns by 73 rows) and ocean (288 by 144) submodels. Each submodel has a separate array of primary data, which is swapped at coupling intervals (usually 1 day). Coupling fields are gathered into a global domain for interpolation to the grid of the new submodel before the integration proceeds. The coupling calculations here are sequential, which impacts scalability, but coupling costs are less than 5% of the total elapsed time for a standard climate run with 72 processors.

An alternative method of coupling using OASIS coupling software developed at CERFACS has been introduced into the UM for coupling externally supplied models without needing to make extensive changes to code. This technique spawns slave processes for atmosphere and ocean models, synchronisation taking place with communication by unix pipes at coupling intervals.

2.5 Input/output

A distinctive feature of the UM system, labelled STASH, is the capability to extract a wide range of model output fields with extra processing inside the model under user control. Output fields of global domain were gathered onto a single processor and output to an expanding file. Most forms of STASH processing involving time-meaning and spatial averaging were dealt with simply in this way. Most effort was required in dealing with the output of lateral boundary conditions for regional models, extraction of subsets of domains and timeseries of fields at individual grid points. In particular, interpolation of fields for generating regional boundary conditions comprised was costly in comparison with sequential code equivalents.

Parallel I/O to a file was not found to be practical. Asynchronous I/O was attempted and used with some success for a number of applications. However in an operational environment, difficulties with closing of files before the next sequence in the suite led to a lack of robustness and this technique has not been adopted for the current system. A number of other I/O changes, including re-structuring of output files in a 'well-formed' format and pre-fetching of files led to significant savings.

2.6 Validation of migrated code

An assumed constraint for the MPP code structure is that model results should exactly compare at the bit level for any number or arrangement (ie N-S:W-E split) of processors, and also against the non-MPP single processor equivalent. This constitutes a strong test of the quality of coding and is a powerful tool in exposing errors. In fact, a small number of minor errors in pre-existing non-MPP code became apparent during the migration phase. A single exception to this constraint lies in code for assimilation of observations, where the associated loss of efficiency would have been too great, and bit reproducibility is achieved with different arrangemens of processors, but only for the same number of processors. The main extra cost lies in the special treatment required for global sums, ensuring that results are added from processors in the same order. It was found necessary to check model data states at the full 64 bit precision of the model - standard UM data files compressed to 32 bit precision masked the onset of departure of results. The constraint of ensuring compatibility between non-MPP and MPP modes for model runs is likely to become less important in the future, once practical model configurations become too big to fit into a single large memory (128 Mbytes) processor.

Model results for code migrated from Cray-C90 to Cray-T3E computers could not be compared exactly because of the change from Cray to IEEE format number representation. A development suite was built up on the Cray-T3E through 1997 to duplicate the operational system. Verification of meteorological variables were compared to ensure that no gross changes had arisen. As development of the new suite progressed, various optimisation changes affecting answers at the bit level were needed to meet timing targets. It was found that virtually all verification comparisons were within the expected tolerance. The only exceptions were small improvements in scores which have been attributed to using 64 bit arithmetic for maths library functions, possibly with the extra precision of IEEE formatted numbers adding further benefit.

Climate modelling studies place a greater reliance on close bit comparability of model results for non-scientific modifications, since even minor systematic changes can lead to a climatological signal. Validation was achieved by running a series of short control forecasts from initial states with minor perturbations. The new model code - migrated to the MPP and optionally including optimisations - was then run to check that its results fell within the control envelope. This was particularly effective when the process was repeated, gradually stepping through individual modules during the first timestep, which helped to identify departures in evolution.

3. Problems encountered during migration

In order to reduce costs, some calculations in halo regions were omitted, often with DO loops missing out the first and last rows in local calculations over the horizontal domain. This led to occasional inconsistencies and uninitialised data culminating in model errors. In particular, the treatment of processors containing polar rows - with a halo beyond the polar row - generated problems for both atmosphere and ocean models. This arose because of an implicit assumption for original code in both models whereby polar values were updated with respect to dynamical but not to physical parametrisation

calculations.

A few portability issues were raised. The increased precision but smaller range of IEEE versus Cray format numbers exposed some conditional tests that were better handled by Fortran90 intrinsics. Fortran NAMELISTs are used extensively in the UM but are not standardised effectively for different compilers, and needed extra effort.

The UM has a fairly complex top level structure of Unix scripts that are required to handle the different and extensive needs of operational and climate production running, including model compilation, reconfiguration of model states, automatic post-processing and a capability for restarting interrupted runs. This proved to be much slower in an MPP environment and required considerable revision of Unix control scripts in order to reduce inter-process communications at the unix level.

I/O costs were a major overhead for the model at the outset of the migration project but also, naturally, varied with the details of system implementation, such as the level of disk striping and I/O cache memory, as progressive upgrades were introduced. Together with the increasing volume of work processed by the machine as more users migrated to the Cray-T3E, each change tended to affect timing bottlenecks and load balancing requirements, making it difficult to identify which areas of code needed the most effort. Also, since potential I/O savings were associated with relatively large code changes within the model, such as for asynchronous I/O, it was difficult to evaluate their effect in advance.

Job scheduling of UM integrations for the variety of Met. Office users was found to be a challenging task. The operating system is still evolving to make the best use of MPP resources for the combination of large operational UM jobs running to deadlines with a set of background climate jobs running large integrations and short term development needing a fast turnaround.

A set of early problems arose from model runs being suspended at a barrier through a conflict in message passing. These occurred intermittently due to the variations in communications traffic from other work on the machine. Errors of this sort were generally more difficult to track down and required extra diagnostics to be included in an alternative message passing interface to identify specific processors and messages. The general quality of proprietary and other diagnostic tools improved during the migration period, which aided problem solving as the MPP model code matured.

4. Current developments

Having met initial operational timing targets, most work on the current UM is now concentrating on single processor optimisation, since many opportunities remain for savings within physics parametrisation codings. Effort on improving load balancing for specific code areas has continued. For example sunlit grid points will be redistributed over processors for short-wave radiation calculations; and a test for convecting points will be made to redistribute convection calculations over processors. Load balancing for filtering in the ocean model is being improved by allocating work dynamically. Division into segments of data - rather than just rows - enhances the balancing process, and it was found that an assumption of work being proportional to the square of segment length provided a simple estimate of processor load.

The main areas of new work lie in the introduction of two major new components into the UM: variational assimilation (VAR) and a new semi-implicit non-hydrostatic dynamics, which are both in advanced development but need to meet stringent accuracy and timing targets. These components have

adopted the same parallelisation paradigm as the rest of the UM system, and will share message passing library interfaces and routines where possible, but have required local solutions in their own applications. In particular, VAR has made extensive use of Fortran90 constructs, which has provided extra tests for MPP capabilities.

Acronyms

- CPU: Central Processing Unit
- GCOM: Generalised Communication package for message passing: a set of Fortran library routines comprising a flexible interface between model code and the chosen method of message passing.
- HADCM3: A specific coupled model developed at the Met. Office Hadley Centre for climate prediction experiments, involving extended production running.
- MPP: Massively Parallel Processing: signifies distributed memory in this context.
- OASIS: Ocean-atmosphere coupling software developed by CERFACS in France.
- STASH: System for processing model output diagnostics from the UM.
- UM: Unified Model.
- VAR: Variational Assimilation.

OPTIMIZING MC2 FOR RISC ARCHITECTURES: FORECAST ACCURACY VERSUS PERFORMANCE

Stephen Thomas, Joshua Hacker, Michel Desgagné, Roland Stull

Recherche en prévision numérique (RPN), Environment Canada,
2121, route Transcanadienne, Dorval, Québec H9P 1J3, CANADA
steve.thomas@ec.gc.ca, michel.desgagne@ec.gc.ca
Tel. 1-514-398-5157, FAX 1-514-398-6115

Atmospheric Science Programme, University of British Columbia
1984 West Mall, Vancouver, B.C., V6T 1Z2, CANADA
jhack@geog.ubc.ca, rstull@geog.ubc.ca
Tel. 1-604-822-2148, FAX 1-604-822-6150

Abstract

Current generation RISC microprocessors operate at clock frequencies ranging up to 1 GHz with the ability to complete two or more floating point operations (flops) per clock cycle. To sustain a significant percentage of peak performance, large secondary L2 cache memories based on fast SRAM technology are essential. Single processor optimisations are presented for the MC2 model code on the MIPS R10000 and SUN UltraSparc II microprocessors. Ensemble forecast techniques for high resolution mesoscale simulations are applied to assess the impact of aggressive floating point optimisations on forecast accuracy. Parallel benchmarks of the MC2 model (adiabatic kernel + physics) on the SGI/Cray Origin 2000 and Fujitsu AP3000 are also presented. The relative efficiency of line relaxation preconditioners for minimal residual Krylov iterative solvers is reported in the context of real-time mesoscale forecasting.

1. Introduction

The performance of RISC microprocessors continues to double approximately every eighteen months according to Moore's law and advances in semiconductor technology have continued unabated for the past 30 years. There is every reason to believe that these trends will continue at least in the short term. The current generation of pipelined RISC processors includes the MIPS R10000, SUN UltraSparc II, DEC Alpha and HP-PA chips. These processors operate at clock frequencies ranging up to 1 GHz with the ability to complete two or more floating point operations (flops) per clock cycle. To sustain a significant percentage of peak performance, code restructuring in combination with compiler and run-time optimisations must be applied judiciously. Sustainable floating point execution rates are to a large extent determined by optimal use of the memory hierarchy. Typically, secondary or level two (L2) cache utilisation must surpass 95% before large performance gains are realized. Stride-1 memory references and cache blocking to promote data locality both serve to increase primary L1 and secondary L2 cache hit ratios. Manual loop unrolling and interchanges will often expose instruction level parallelism to the compiler. Given information about the memory hierarchy, a compiler can often schedule instructions for pipelined execution and optimal cache usage. More aggressive optimisations include floating point instruction re-ordering and reduced precision in math libraries in exchange for increased speed. The most aggressive optimisations must be applied with care as they may seriously degrade the accuracy of a fluid flow simulation. It is for this reason that we believe ensemble analysis techniques can be useful in assessing the impact of optimisation strategies on atmospheric models which include complex physical parametrisation packages.

RISC microprocessors, high-speed SRAM caches and high-bandwidth interconnection networks form the building blocks of modern distributed and distributed-shared memory parallel computer architectures. In this paper we examine the performance of a high-resolution mesoscale limited-area atmospheric model on the SGI/Cray Origin 2000 and Fujitsu AP3000 parallel computers using the Message-Passing Interface (MPI). The Canadian MC2 is a fully compressible nonhydrostatic model based on second-order finite differences in space and a three-time-level semi-implicit, semi-Lagrangian time discretisation. The use of a terrain-following height coordinate results in a highly nonsymmetric linear system to solve every time step for the pressure. Minimal residual Krylov solvers have thus been implemented and the computational efficiency of the model using line relaxation preconditioners designed for both hydrostatic and nonhydrostatic flow regimes is presented.

2. MC2 Model Formulation

The Mesoscale Compressible Community (MC2) model is a fully compressible nonhydrostatic limited area atmospheric model used in Canadian Universities and Environment Canada for mesoscale and microscale atmospheric research. A detailed description of the adiabatic kernel and numerical formulation of the MC2 model with open boundaries is given in Thomas et al. [9]. In particular, the model employs a semi-implicit semi-Lagrangian time discretisation scheme and a non-orthogonal coordinate system based on the terrain-following transformation

$$Z(X,Y,z) = H\left[\frac{z-h(X,Y)}{H-h(X,Y)}\right]$$

introduced by Gal-Chen and Sommerville [2], where $h(X,Y)$ is the height of topography. Following standard conventions, the Jacobian G and metric coefficients G^{IJ} of the transformation are denoted

$$G = \frac{H-h}{H}, \qquad G^{13} = \frac{1}{G}\left[\frac{Z-h}{H}\right]\frac{\partial h}{\partial X}, \qquad G^{23} = \frac{1}{G}\left[\frac{Z-h}{H}\right]\frac{\partial h}{\partial Y}$$

Given a polar stereographic projection at reference latitude ϕ_0 with map factor $m = (1+\sin\phi_0)/(1+\sin\phi)$, $S = m^2$ and Coriolis parameter $f = 2\Omega\sin\phi$, the compressible governing equations in projected $\mathbf{X} = (X,Y,z)$ coordinates become

$$\begin{aligned}
\frac{DU}{Dt} &= fV - K\frac{\partial S}{\partial X} - RT\left[\frac{\partial q}{\partial X} + G^{13}\frac{\partial q}{\partial Z}\right] + F_U \\
\frac{DV}{Dt} &= -fU - K\frac{\partial S}{\partial Y} - RT\left[\frac{\partial q}{\partial Y} + G^{23}\frac{\partial q}{\partial Z}\right] + F_V \\
\frac{Dw}{Dt} &= -g - \frac{RT}{G}\frac{\partial q}{\partial Z} + F_w \\
\frac{DT}{Dt} - \frac{RT}{c_p}\frac{Dq}{Dt} &= \frac{Q}{c_p} \\
\frac{c_v}{c_p}\frac{Dq}{Dt} &= -S\left[\frac{1}{G}\frac{\partial GU}{\partial X} + \frac{1}{G}\frac{\partial GV}{\partial Y}\right] - \frac{1}{G}\frac{\partial GW}{\partial Z} + \frac{Q}{c_pT}
\end{aligned} \tag{1}$$

The contravariant vertical velocity W is related to the covariant velocity components by the equation, $W = G^{-1}w + S(\,G^{13}U + G^{23}V\,)$. Potential temperature is $\Theta = Te^{-\kappa q}$, and $\Pi = (p/p_0)^\kappa$ is the Exner function, where $T = \Pi\Theta$, $q = \ln(p/p_0)$, $p_0 = 1000$ mb. R and c_p are the gas constant and heat capacity for dry air at constant pressure, $\kappa = R/c_p$. U, V and w are the wind images in projected (X,Y,z) coordinates and g is the gravitational

acceleration. $K = (U^2 + V^2)/2$ is the pseudo kinetic energy per unit mass. Momentum (F_U, F_V, F_w) and heat Q sources or sinks are also included.

The semi-implicit scheme results in an elliptic problem to solve every time step for a log pressure perturbation q' about a stationary isothermal hydrostatic basic state. The nonsymmetric system of equations resulting from a finite difference discretisation is solved using the Generalized Minimal Residual (GMRES) algorithm of Saad and Schultz [3]. Skamarock et al. [4] use the mathematically equivalent GCR algorithm of Eisenstat et al. [1] in a semi-implicit formulation of a compressible model (see also Smolarkiewicz and Margolin [5, 6], Smolarkiewicz et al. [7]). The solver convergence criteria is based on the RMS divergence or an estimate thereof, since it indicates when the discrete form of the Gauss divergence theorem with open boundaries has been satisfied. Krylov subspace methods are particularly well-suited to a distributed memory, message passing model of computation since they rely primarily on distributed matrix vector multiplication and an inner product implemented as a global reduction summation. Computational efficiency (overall wall-clock time) is improved by finding a suitable preconditioner, which is often problem dependent. The discretised elliptic operator in a nonhydrostatic pressure solver will be dominated by the vertical terms when the aspect ratio $\Delta X/\Delta Z$ is large. Therefore, an effective preconditioning strategy is to invert the vertical components of the elliptic operator and Skamarock et al. [4] apply a vertical alternating direction implicit (ADI) line relaxation preconditioner.

A vertical ADI line relaxation preconditioner for the $n \times n$ linear system $Ax = b$ is based on the splitting $A = H + V$, where the H and V represent the horizontal and vertical components of the discrete elliptic operator based on centered second-order finite differences. The ADI iteration is derived from the pseudo-time integration of the heat equation $u_t = \Delta u + r$ to steady state, where the matrix A represents the discrete Laplacian and

$$(I - \beta V)x^{k+1} = (I + \beta H)x^k - \beta b \tag{2}$$

The largest possible pseudo-time step β is chosen so that the above integration scheme remains stable. A slightly more implicit scheme can be constructed using a line Jacobi relaxation scheme

$$Tx_i^{k+1} = x_{i+1}^k + x_{i-1}^k + b, \qquad i = 1, \ldots n \tag{3}$$

where the index i represents an entire line of grid points. For the Poisson problem $-\Delta u = r$ on the 2D unit square, the matrix $T = \mathrm{diag}(A_{i+1,i}, A_{ii}, A_{i-1,i})$ is block tridiagonal, where $A_{i+1,i} = A_{i-1,i}$, $A_{i-1,i}$ and $A_{ii} = T_i$. Second-order centered finite differences imply $T_i = \mathrm{diag}(-1, 4, -1)$. The vertical ADI scheme (2) splits and weights (with β)

the diagonal terms of the discrete operator, whereas the line Jacobi scheme inverts the diagonal and vertical off-diagonal terms of the operator. For nonhydrostatic problems on isotropic grids, a fully 3D ADI preconditioner is implemented as in Skamarock et al. [4]. The solution of tridiagonal linear systems of equations implies global data dependencies, thus a parallel data transposition strategy has been adopted in a distributed-memory implementation of the 3D ADI preconditioner.

3. Parallel Performance

For a distributed-memory SPMD model of parallel computation, the $N_i \times N_j \times N_k$ computational grid is partitioned across a $P_X \times P_Y$ logical processor mesh. A domain decomposition in the horizontal direction is employed due to the strong vertical coupling in physical parametrisation packages and since the number of grid points in the vertical direction is typically one order of magnitude less than in the horizontal. Each processor therefore contains $N_i/P_X \times N_j/P_Y \times N_k$ points, resulting in a near optimal surface to volume grid point ratio for semi-Lagrangian advection and application of the elliptic operator in the GMRES solver [8]. For both algorithms the communication overhead associated with boundary data exchanges between subdomains is minimal when compared with computations. The 1D vertical ADI and Jacobi line relaxation preconditioners are also well-suited to a horizontal decomposition, since the only global data dependency is in the vertical direction within tridiagonal solvers. However, the 3D ADI preconditioner requires global data in each of the three coordinate directions in order to solve tridiagonal linear systems of equations during each ADI sweep. Thus, the right-hand side b and solution x^k must be re-mapped to perform line relaxations in each coordinate direction in turn. Such a re-mapping takes the form of a data transposition algorithm requiring collective MPI all-to-all communication of $\mathcal{O}(N^3)$ grid points. Vertical sweeps in the Z direction are performed using the domain decomposition described above. Each processor contains $N_i \times N_j/P_Y \times N_k/P_X$ grid points for sweeps in the X direction and $N_i/P_Y \times N_j \times N_k/P_X$ points in the Y direction. ADI sweeps progress from left to right and then right to left as indicated below with arrows representing communication steps.

$$\frac{N_i}{P_X} \times \frac{N_j}{P_Y} \times N_k \quad \Longleftrightarrow \quad N_i \times \frac{N_j}{P_Y} \times \frac{N_k}{P_X} \quad \Longleftrightarrow \quad \frac{N_i}{P_Y} \times N_j \times \frac{N_k}{P_X}$$

To compare the computational efficiency of the model using the parallel 1D Jacobi and 3D ADI line relaxation schemes, a quasi-hydrostatic test case was run on an SGI/Cray Origin 2000 computer with sixteen 195 MHz R10000 processors each containing a 4 MB L2 cache.

The purpose of our test was to determine if the communication overhead associated with the data transposition strategy would adversely affect the computational efficiency as the number of processors is increased. A $120 \times 120 \times 35$ grid at 2.5 km horizontal resolution with model lid set at 23 km was employed in a mesoscale forecast over the British Columbia lower mainland. The 30 hour forecast using the MC2 model was run with version 3.5 of the Recherche en prévision numérique (RPN) physics package including radiation and stratiform condensation parametrisations. The integration consisted of 1800 time steps of length $\Delta t = 60$ sec using both 1D and 3D preconditioners and the results are summarized in Table 1 using single processor optimisation level 3 (described in section 4).

$P_X \times P_Y$	1×1	2×2	2×4	3×4	4×4
1D Jacobi	29:09	7:18	3:34	2:37	2:00
3D ADI	32:51	8:13	3:38	2:42	2:19

Table 1: MC2 execution time on SGI/Cray Origin 2000 (hrs:mins)

Despite the fact that the 3D ADI preconditioner results in a much faster convergence rate for the GMRES solver, the overall model execution times are very close. Since the grid aspect ratio $\Delta X/\Delta Z$ for this problem is $\mathcal{O}(10)$, the 1D line Jacobi scheme is still competitive. Moreover, the data transposition overhead appears not to adversely affect performance up to 16 processors.

4. Single Processor Optimisations

To assess the impact of aggressive compiler optimisations on forecast accuracy, three optimisation levels were identified for version 7.1 of the SGI MIPSpro compiler and linker, targeted for version IP27 of the MIPS R10000 microprocessor. These compiler options were tested with 195 MHz and 250 MHz versions of the processor on an Origin 2000 computer equipped with either 1MB or 4MB L2 caches.

Optimisation Level 1 (O1)

```
FFLAGS = -n32 -mips4 -r10000

% mpif77 -lm -lblas -o prog.f
% mpirun -np 4 mc2.Abs
```

Optimisation Level 2 (O2)

```
FFLAGS = -n32 -mips4 -r10000 -O2 -align32 \
-TARG:platform=ip27:processor=r10000 \
-IPA:alias=ON:addressing=ON:opt_alias=ON \
-LNO:opt=1:optimize_cache=1:cs1=32K:ls1=32:assoc1=2:mp1=10:cs2=1M:ls2=128:\
assoc2=2:mp2=10:cs3=512M:ls3=0:is_mem3=ON:cs4=0 \
-OPT:pad_common=ON:inline_intrinsics=ON:cray_ivdep=TRUE

% mpif77 -IPA -multigot -lm -lblas -o prog.f
% fpmode performance fpmode spec mpirun -np 4 mc2.Abs
```

Optimisation Level 3 (O3)

```
FFLAGS = -n32 -mips4 -r10000 -O3 -align32 \
-TARG:platform=ip27:processor=r10000 \
-IPA:alias=ON:addressing=ON:opt_alias=ON \
-LNO:opt=1:optimize_cache=1:cs1=32K:ls1=32:assoc1=2:mp1=10:cs2=1M:ls2=128:\
assoc2=2:mp2=10:cs3=512M:ls3=0:is_mem3=ON:cs4=0 \
-OPT:pad_common=ON:inline_intrinsics=ON:cray_ivdep=TRUE

% mpif77 -IPA -multigot -lfastm -lblas -o prog.f
% fpmode performance fpmode spec mpirun -np 4 mc2.Abs
```

A small number of manual code optimisations are applied to promote stride-1 access and cache-blocking in the adiabatic kernel of MC2. In addition, optimised BLAS subroutine libraries were specified at load time. In-line directives and options were applied to the physical parametrisation package to handle aliasing and loop dependencies assumed by the compiler in the case of some dynamically allocated arrays. The first optimisation level (O1) instructs the compiler to use 32-bit arithmetic with the MIPS 4 instruction set on the R10000 processor. The next level of optimisation (O2) permits a restricted amount of floating-point instruction re-ordering, 32-bit alignment, common block padding to prevent cache thrashing and explicitly defines the memory hierarchy (cache size, line size, associativity) of the target machine to aid in instruction scheduling. Interprocedural analysis is applied to improve both instruction and data cache usage through code movement. The most aggressive level of optimisation (O3) permits floating-point instruction re-orderings that could adversely affect the precision of certain computations such as di-

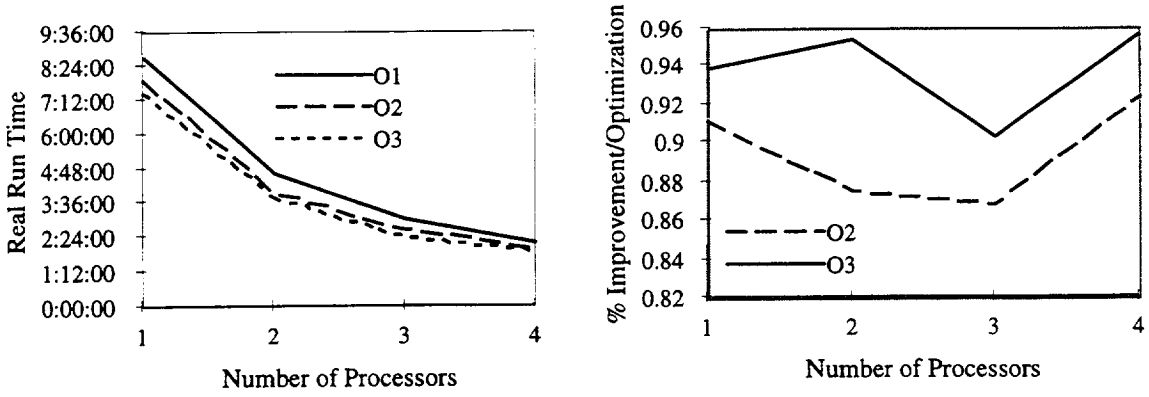


Figure 1: Real run times for different optimization levels and processor configurations. CPU times are shown in (a), and the improvement over the lowest optimization level (O1) is shown in (b).

vision. Fast math libraries can also be employed and here again precision may be reduced in exchange for speed.

In order to evaluate these optimisations in the context of a real-time mesoscale forecasting system, the computational domain and related run-time parameters were taken from daily runs of the University of British Columbia (UBC) ensemble forecasting system (http://spirit.geog.ubc.ca/~model). Horizontal resolution is 10km at 60°N with the model lid at 19km. The grid contains 120×70 points on 35 terrain-following levels, implying the run is quasi hydrostatic with $\Delta X/\Delta Z = 20$. Vertical line Jacobi preconditioning is therefore applied in all our tests. Initial and boundary conditions are obtained from a coarse grid ($\Delta X = 30$km) MC2 run starting at 00UTC 27 November 1997. A 42 hour integration requires 1260 time steps of length $\Delta t = 120$sec. A forecast initialized 06UTC 27 November 1997 was chosen as a benchmark since it is a representative mesoscale case for the British Columbia lower mainland. In particular, it exhibits typical fall and winter characteristics, including moist, south westerly flow at low levels, a persistent upper level trough offshore, and dynamically forced precipitation enhanced by steep topography. The dynamic forcing came from a weak surface cold frontal passage supported by a short wave trough aloft. Upstream boundary wind speeds were on the order of 10ms^{-1} at the surface.

Wall-clock execution times (including input/output) are shown in Figure 1a and performance data for each optimisation level is presented in Tables 2 – 4. The machine

configuration is an Origin 2000 with four 195 MHz R10000 processors equipped with 1 MB L2 cache. Flop counts are derived from hardware counters available on a NEC SX-4 vector supercomputer at the Canadian Meteorological Center (CMC). Note in particular that in all cases the L2 cache utilisation is below 90% and that single processor performance does not exceed 65 Mflops/sec. Figure 1b displays the percentage improvement in wall-clock time (up to four processors) for optimisation levels O2 and O3 compared to level O1. The plot confirms that optimization levels O2 and O3 produce similar improvements for all processor configurations.

$P_X \times P_Y$	1×1	1×2	1×3	2×2
Wallclock	31395	16666	11169	8034
User	30975	16339	10828	7879
MFLOPS/sec	53.6	101.5	153.2	210.6
MFLOPS/sec/PE	53.6	50.8	51.1	52.6
L2 hit rate	0.843	0.880	0.881	0.892

Table 2: Optimisation Level O1, 195 MHz R10000 + 1 Mb L2 cache.

$P_X \times P_Y$	1×1	1×2	1×3	2×2
Wallclock	28597	14593	9707	7420
User	28223	14341	9449	7252
MFLOPS/sec	58.8	115.7	175.6	228.8
MFLOPS/sec/PE	58.8	57.8	58.5	57.2
L2 hit rate	0.845	0.886	0.893	0.891

Table 3: Optimisation Level O2, 195 MHz R10000 + 1 Mb L2 cache.

$P_X \times P_Y$	1×1	1×2	1×3	2×2
Wallclock	26862	13919	8769	7099
User	26828	13679	8578	6935
MFLOPS/sec	61.8	121.3	193.4	239.2
MFLOPS/sec/PE	61.8	60.6	64.5	59.8
L2 hit rate	0.838	0.885	0.893	0.894

Table 4: Optimisation Level O3, 195 MHz R10000 + 1 Mb L2 cache.

$P_X \times P_Y$	1×1	1×2	1×3	2×2	2×4	3×4
Wallclock	21360	10813	7165	5416	2841	2035
User	21180	10740	7051	5344	2744	1949
MFLOPS/sec	78.3	154.5	235.3	310.4	604.6	851.2
MFLOPS/sec/PE	78.3	77.2	78.4	76.2	75.6	70.9

Table 5: Optimisation Level O3, 195 MHz R10000 + 4 Mb L2 cache

The benchmark case described above has also been run on an Origin 2000 with up to sixteen 195 MHz R10000 processors configured with 4 MB L2 cache and the results are reported in Table 5. A significant increase in the cache utilisation (over 97%) is obtained when moving to the 4 MB L2 cache and per processor performance increases accordingly to just under 80 Mflops/sec or 20% of peak when using optimisation level O3. With the recently released 250 MHz R10000 chip, a further 20% improvement in per processor performance has been observed in our benchmarks. In particular, MC2 achieves over 105 Mflops/sec on this processor in combination with the 4 MB L2 cache, although L2 cache utilisation drops back down to 92%. For comparison, the 10km benchmark was run on a Fujitsu AP3000 with 2 processors per SMP node. This machine is based on the 300 MHz SUN UltraSparc II processor with 2 MB L2 cache, a proprietary high-speed memory architecture and 2D torus interconnect. Compiler options are given below (including the memory hierarchy specification) and performance data is summarized in Table 6.

```
FFLAGS = -O4 -dalign -xarch=v8plusa -xchip=ultra \
-fsimple=2 -xdepend -xlibmil -xlibmopt -xsafe=mem \
-xcache=16/32/4:2048/32/1
```

$P_X \times P_Y$	1×1	1×2	1×3	2×2	2×4
Wallclock	25163	12706	8820	6905	4054
User	24853	11986	8071	6152	3023
MFLOPS/sec	66.8	138.4	205.6	270.0	548.8
MFLOPS/sec/PE	66.8	69.2	68.5	67.5	68.6

Table 6: Fujitsu AP3000, 300 MHz AP3000 + 2MB L2 cache

5. Meteorological Results

a. Evaluation Methods

The effect of RISC compiler and run-time optimizations on the accuracy of high resolution mesoscale forecasts is still a largely unexplored subject. One possible way of studying these effects is to generate an ensemble of forecast runs, regarding optimisation as a possible source of errors or perturbations. Given three optimisation levels and four processors, a simple way to produce an ensemble is to run the same meteorological test case 12 times, varying optimization levels and processor configurations, as described above. The 12 runs comprise an ensemble and standard analysis techniques can reveal error growth and forecast spread as a function of optimization and floating-point instruction re-ordering.

Forecast spread, rather than error values, will be emphasized for two reasons. First, a control run to characterize deterministic model error is not available. In ensemble forecasting, a control run is often the categorical forecast, initialised with a best guess analysis and perturbing this analysis gives initial conditions for the other ensemble members. Ensemble forecasters then look for forecast spread and a reduced ensemble-averaged error. Looking for error reduction will not provide insight into the problem at hand. Although compiler optimisations can be considered as perturbations, the analogy should not be extended too far since the initial conditions are not perturbed. A single processor run without any optimization could be taken as the control run since it may be what the model developers intended, but given codes of such complexity this is difficult to quantify. Second, one case study can not determine the statistical properties of the ensemble in a robust sense. Thus, it is unwise to select a top performer in this experiment, or to average the results looking for an improvement. However, significant deviation of one group of forecasts from the others indicates that caution is warranted.

Each ensemble member is initialized identically and common boundary conditions are applied throughout the 42h integration period. The 00h forecasts deviate from each other slightly because grid staggering and destaggering require calculations that are handled differently depending on optimization and processor configuration. Boundary value determination is subject to the same errors and by examining the spatial distribution of forecast spread such effects can be quantified. To evaluate the error in each run, model forecasts of temperature, mean sea level pressure (MSLP) and 12h accumulated precipitation are compared with surface observations from weather stations operated by the Canadian Atmospheric Environment Service (AES). Observation error was not consid-

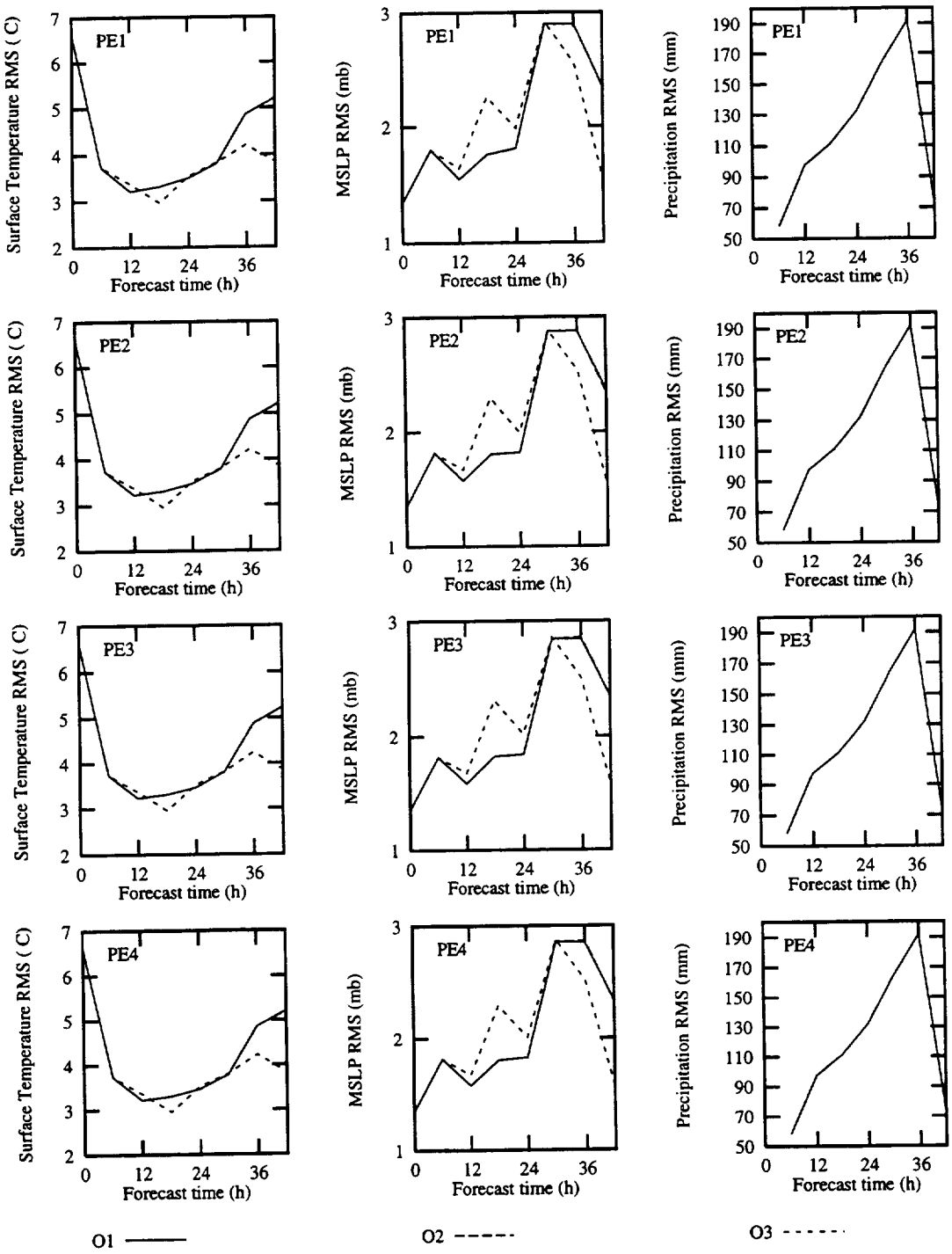


Figure 2: RMS error at each forecast hour for surface temperature (column 1), MSLP (column 2) and 12h accumulated precipitation (column 3). The legend indicates curves for 3 optimization levels and the number of processors employed is shown on each plot.

ered and the observations are subject to AES quality standards. A cubic polynomial is used to interpolate temperature, MSLP, and precipitation values from the model grid to observation sites. Errors are evaluated by averaging over every available station site at each 6h interval. At least 50 stations were available for verification at these times. RMS results are therefore the average RMS over all the stations reporting within the model domain. Although the model domain is three dimensional, a surface comparison can identify significant discrepancies in the forecasts. The results will not provide information about spatial distribution of errors, but will show the error differences and thus average forecast differences.

b. Ensemble Results

Results for the entire ensemble are shown in Figure 2. It is clear that at forecast times greater than 6 hours, every run with full optimization (O3) significantly deviates from the rest of the ensemble with respect to MSLP and temperature. Because error spreads between different processor configurations are too small to observe on these plots, each is shown on an individual plot. These are negligible compared to the deviations present in the O3 runs. The decrease in temperature error over the first 6h of forecast is a result of the improved resolution from the initial conditions specified at 30km to the 10km model domain. Precipitation spread is negligible when compared to the total model error. The maximum at 36h is a property of only this forecast and would disappear with several forecasts over different time periods.

The spatial distribution of forecast spread can be understood by looking at the station and gridded results. Figure 3a and b show the maximum temperature and MSLP differences between any two forecasts after 42h of integration, interpolated to station sites. Temperature differences range from 1°C near the upstream boundary to 4°C near the downstream boundary, with a general increasing trend from west to east. A similar trend is evident in the MSLP differences, which range from 0mb in the southwest to 4mb in the northeast part of the domain. Thus, numerical errors accumulate for air parcels with longer residence times in the domain.

Figures 3c and d, show the difference in gridded fields between ensemble members 1 (O1PE1) and 12 (O3PE4), also after 42h of integration. For comparison with Figures 3a and b, surface temperature and MSLP differences are shown. These two members are responsible for many (but not all) of the maximum differences at station sites. The increasing differences toward the downstream boundary are clear. It is worth noting that the spatial distribution of the differences does not show this property at the 00h forecast time. Rather, the differences are evenly distributed and resemble numerical noise. The

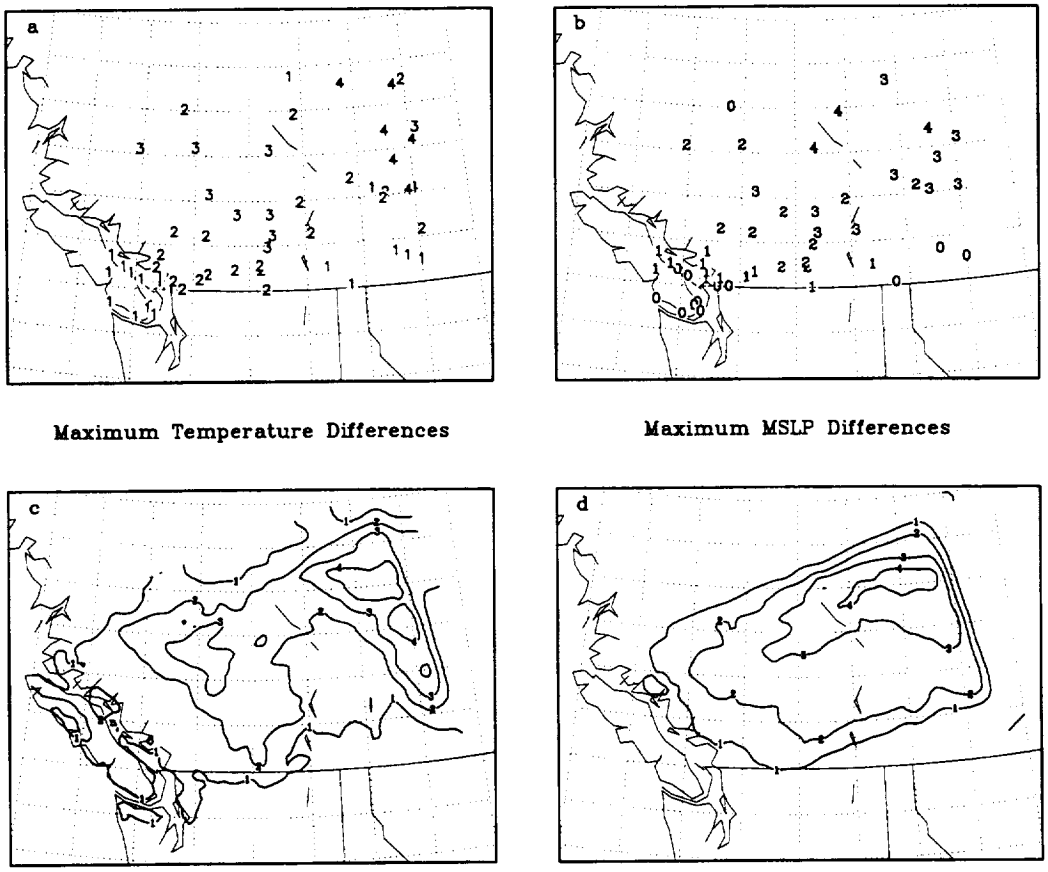


Figure 3: Spread between ensemble members. The maximum forecast spread in surface temperature (°C) (a) and MSLP (mb) (b). Gridded spreads between ensemble members 1 (O1PE1) and 12 (O3PE4) for temperature (c) and MSLP (d) are also shown.

features shown here develop through the forecast period. The differences in the gridded fields approach zero at all the boundaries except the outflow boundary. This is evidence that the boundary formulation is not significantly affected by the different optimization levels and processor configurations. Error must have time to accumulate during the integration and the common boundary conditions result in small boundary differences even at the end of the forecast.

6. Conclusions

Our performance studies indicate that fully 3D implicit methods based on minimal residual Krylov iterative solvers and line relaxation preconditioners work well on distributed-shared memory architectures such as the SGI/Cray Origin 2000. Perhaps the best way to evaluate the impact of code optimisation strategies for RISC microprocessors on atmospheric flow simulations is by using theoretical test cases with analytic or known solutions. In such cases, a loss of precision can be isolated and easily corrected. With a fully configured atmospheric model interfaced to complex physical parametrisation packages, the task is more difficult. One possible approach is to view aggressive compiler optimisations as a possible source of floating point errors or perturbations of the initial and boundary conditions. Ensemble forecast techniques can then be applied to evaluate if a trade-off between accuracy and performance is within acceptable bounds.

It must be emphasized that it is impossible to know *a priori* which level of optimization is most accurate since this is code and platform dependent. However, significant deviations over several forecasts indicates caution should be exercised when applying optimizations. Such effects should be quantified and closely examined in either an operational or research environment. The results shown in Figure 2 and Figure 3 suggest that compiler and run-time optimization can have a significant impact on short range forecasts. RMS errors can deviate by more than 25% of the total error for surface temperature and MSLP. The O3 error is at times less than the O1 and O2 error and thus the only way to characterize accuracy is by running similar ensembles over many forecast periods. Obtaining a 'best' forecast from one case is unwise. Error spreads of similar magnitude will likely occur in many other meteorological cases, but the value of the error and the rank of each ensemble member will vary.

Forecasting implications of these differences should not be ignored. Both the MSLP and surface temperature differences are large. For example, four degree temperature differences are well above the accuracy required to determine precipitation phase (solid or

liquid) in many instances. This can translate to thousands of dollars wasted by poor planning for industry, agriculture and government agencies that depend on accurate weather information. A 4 mb MSLP difference will affect wind forecasts, which are equally as important. Although only surface impact is addressed here, the dependency of surface parameters on the upper air conditions suggest effects aloft are just as great. Furthermore, the nonlinear nature of the atmosphere and further accumulation of optimization errors will cause the differences in forecasts to grow with forecast duration. Thus, similar analyses of medium range forecasts (3-10 days) would show a much larger spread between forecasts, due to compiler optimization.

Acknowledgements

The authors would like to thank Piotr Smolarkiewicz from NCAR/MMM for his interest and encouragement to find better preconditioners for minimal residual Krylov solvers. An A-grid version of a line Jacobi scheme was first tested in the EULAG model [6]. In the near future we also hope to benchmark MC2 on the NCAR HP/Exemplar SPP 2000 DSM with support from Steve Hammond and SCD. We would also like to thank SGI, Fujitsu and the Canadian Meteorlogical Center for providing dedicated computer time.

References

[1] S. C. Eisenstat, H. C. Elman, and M. H. Schultz. Variational iterative methods for nonsymmetric systems of linear equations. *SIAM J. Numer. Anal.*, **2** (1983), pp. 345–357.

[2] T. Gal-Chen and R. C. Sommerville. On the use of a coordinate transformation for the solution of the Navier-Stokes equations. *J. Comp. Phys*, **17** (1975), pp. 209-228.

[3] Y. Saad and M. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput*, **7** (1986), pp. 856-869.

[4] W. C. Skamarock, P. K. Smolarkiewicz and J. B. Klemp. Preconditioned conjugate-residual solvers for Helmholtz equations in nonhydrostatic models. *Mon. Wea. Rev.*, **125** (1997), pp. 587-599.

[5] P. K. Smolarkiewicz and L. G. Margolin. Variational solver for elliptic problems in atmospheric flows. *Appl. Math and Comp. Sci.*, **4** (1994), pp. 527-551.

[6] P. K. Smolarkiewicz and L. G. Margolin. On forward-in-time differencing for fluids: An Eulerian/semi-Lagrangian nonhydrostatic model for stratified flows. *Atmos. Ocean.* , Special Vol. XXXV, no. 1, (1997), pp. 127-152.

[7] P. K. Smolarkiewicz, V. Grubišić and L. G. Margolin. On forward-in-time differencing for fluids: Stopping criteria for iterative solutions of anelastic pressure equations. *Mon. Wea. Rev.*, **125** (1997), pp. 647-654.

[8] S. J. Thomas, A. V. Malevsky, M. Desgagné, R. Benoit, P. Pellerin and M. Valin. Massively parallel implementation of the mesoscale compressible community model. *Parallel Computing*, **23** (1997), 2143-2160.

[9] S. J. Thomas, C. Girard, R. Benoit, M. Desgagné and P. Pellerin. A new adiabatic kernel for the MC2 model. *Atmos. Ocean.*, to appear (1998).

Author: Giri Chukkapalli
San Diego Supercomputer Center
P.O. Box 85608
San Diego, CA 92186-5608
giri@sdsc.edu

A Theoretical and Experimental Analysis of Parallel Complexity of Weather and Climate Algorithms using the Shallow Water Benchmark Suite

In this paper, we evaluate the parallel complexity of weather and climate algorithms with the help of the governing equations and the initial-boundary conditions (I.B.V.P). We define a true measure of simulation efficiency which encompasses various partial measures of efficiencies such as parallel speedups. With this definition in mind, we evaluate the advantages and disadvantages of various spatial and temporal discretization schemes in mapping the weather and climate dynamics onto the massively parallel computers. Using this analysis, a conservative, semi-implicit, weak Lagrange-Galerkin (WLG) Finite Element (FE) scheme on unstructured spherical triangular grid is developed to solve the well-accepted shallow water benchmark problem. Results from various benchmark tests show that the current algorithm is sequentially efficient and accurate. The code is parallelized both in shared memory (P4) and message passing paradigm (MPI). The parallel computation partitioning strategy is designed to achieve good load balance and minimize interprocess communication. By overlapping communications and computations, the communication latencies resulting from the Lagrange-Galerkin algorithm are minimized, thereby achieving good parallel speedups. Results from a wide range of parallel platforms including Cray T3E(260), HP Exemplar(256), IBM SP2(128), Berkeley NOW(100), SGI power, challenge(32) and KSR1(32) show that the present parallel algorithm is efficient, portable, and scalable. Using a timeline analysis of the present algorithm, we identify the reasons for parallel performance degradation and provide possible remedies for minimizing it. We address the design issues that need to be considered for the code to be efficiently portable.

A Scalable Version of the Navy Operational Global Atmospheric Prediction System Spectral Forecast Model

Thomas E. Rosmond
Naval Research Laboratory, Monterey, California
7 Grace Hopper Ave
Stop 2
Monterey, Ca 93943-5502
rosmond@nrlmry.navy.mil
Voice: +1 408 656-4736
Fax: +1 408 656-4769

1. Introduction:

The navy operational global atmospheric prediction system (NOGAPS) is the heart of the Fleet Numerical Meteorological and Oceanographic Center (FNMOC) operational NWP support to all branches of the Department of Defense. The Naval Research Laboratory (NRL) is responsible for NOGAPS design and computer implementation. NOGAPS has been operational at FNMOC since 1982 and has been through several computer system upgrades and design changes during that time. The spectral forecast model component of NOGAPS [1] is similar in formulation to global models run at other major operational NWP centers around the world. Operationally it runs multitasked on a Cray C90 using 10-15 processors with a sustained performance of 400 Mflops/processor. The operational resolution is currently T159L30. In addition to the operational application NOGAPS is run by NRL scientists at a variety of lower resolutions for coupled atmosphere/ocean modeling research, data assimilation studies, long-term integrations such as AMIP, and singular vector/adjoint model research.

Price/performance considerations are driving many supercomputer applications away from expensive vector architectures and toward scalable architectures built around commodity-based components. Numerical weather prediction models such as the NOGAPS spectral forecast model is an example of such an application. FNMOC is currently planning a switch to a scalable architecture for their primary computational resource over the next 2-3 years, and NOGAPS is the most prominent application to be ported to the new system. In anticipation of a new operational platform for NOGAPS, a distributed memory NOGAPS based on message passing (MPI) has been developed and is being tested and optimized. In part 2 of this report the design criteria and priorities of the new code are discussed. Part 3 describes the design of the computationally intensive spherical harmonic transforms. Part 4 discusses some overall model performance issues and load balance problems. Part 5 presents some conclusions, lessons learned, and future plans.

2. Design objectives:

Because of uncertainty in the commercial marketplace for the new architectures, portability among candidate systems is a high priority in the new code. Single node performance is also being emphasized because diabatic processes dominate the computational cost of NOGAPS, and these are embarrassingly parallel. An important consideration of the new code is to retain as much of the current C90 vector code as possible, for two reasons: (a) we do not want to recode the 30-40 thousand lines of code that make up the model's diabatic processes, and (b) multiple processor shared memory "nodes" are likely to be part of many next- generation systems, and existing parallel vector codes should port gracefully to them. Specifically, this means we preserve the "long-vector" legacy of the past as much as possible, although the code has runtime granularity factors that allow control of actual on-processor array sizes and loop lengths. We believe the potential performance penalties this strategy will impose on cache-based processors will be minor.

The first application of the scalable NOGAPS will be as a benchmark code for a FNMOC procurement. Therefore portability across a wide spectrum of potential platforms is essential. Message passing (MPI) is the obvious choice to maximize this portability. The proposed OpenMP standard for on-node shared memory architectures is being anticipated, but not yet implemented. The ultimate goal is to have a single code which can run as a pure MPI application on a single processor/node MPP platform, a hybrid MPI/OpenMP application on a distributed shared memory system, or as a purely shared memory application similar to the current C90 parallel/vector code. The main motivation for this is configuration management; we cannot maintain separate NOGAPS codes for three different architectures. Some overhead in computational cost and memory is inevitable with such a generalized code, but we are prepared to accept this.

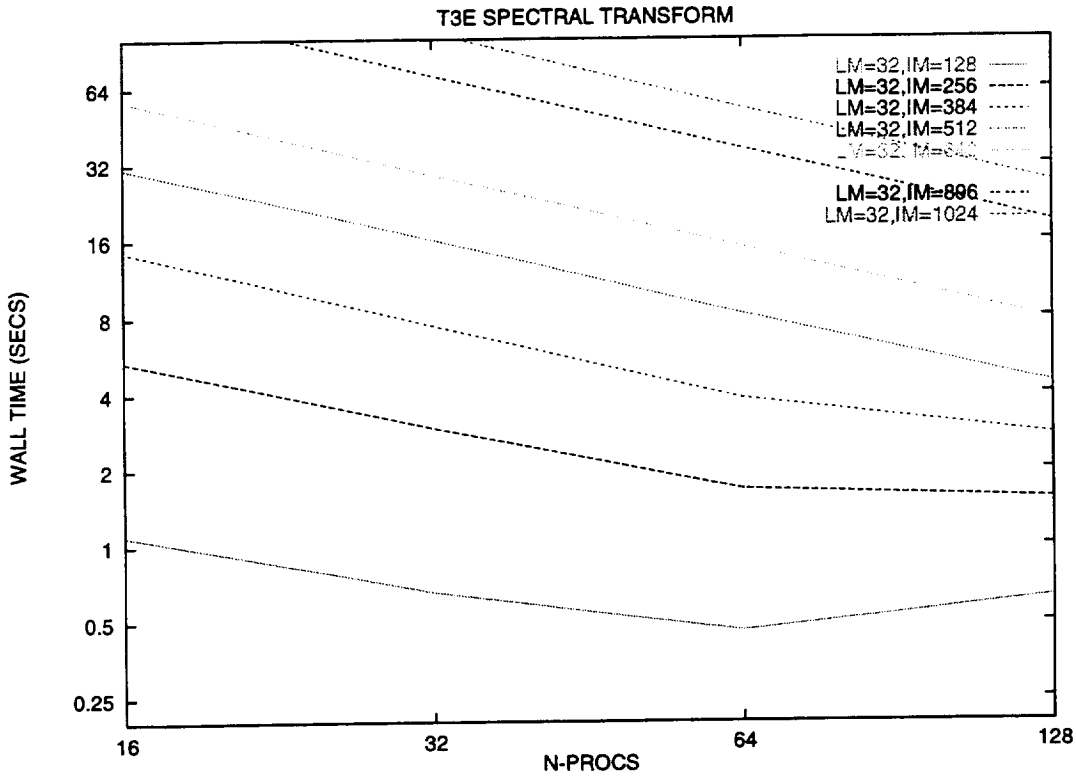


Figure 1: Spherical harmonic transforms on the T3E, showing scaling performance for a variety of spectral resolutions and processor numbers.

3. Spherical harmonic transforms:

Other authors [2, 3] have described parallel versions of the spherical harmonic transform. A common approach is the transpose method, where all communication is confined to matrix transposes that organize data so that all computation can be "on-processor", ensuring bit reproducibility of results for varying numbers of processors,

a vital property for model validation and debugging. The NOGAPS transforms are similar in design to those of other groups, but we have coded them with several different MPI communication modes to allow performance comparisons on a variety of platforms. Specifically we have compared explicit send/recv matrix transposes using combinations of blocking, unblocking, synchronous, and non- synchronous MPI, as well as the mpi_alltoall collective function. Fig. 1 shows some of these results for a range of spectral resolutions and processor numbers on the Cray T3E. Fig. 2 shows a breakdown for one spectral resolution (T213), showing how the FFT, Legendre transform (BLAS matrix multiply), and communication scale for the same range of processor numbers.

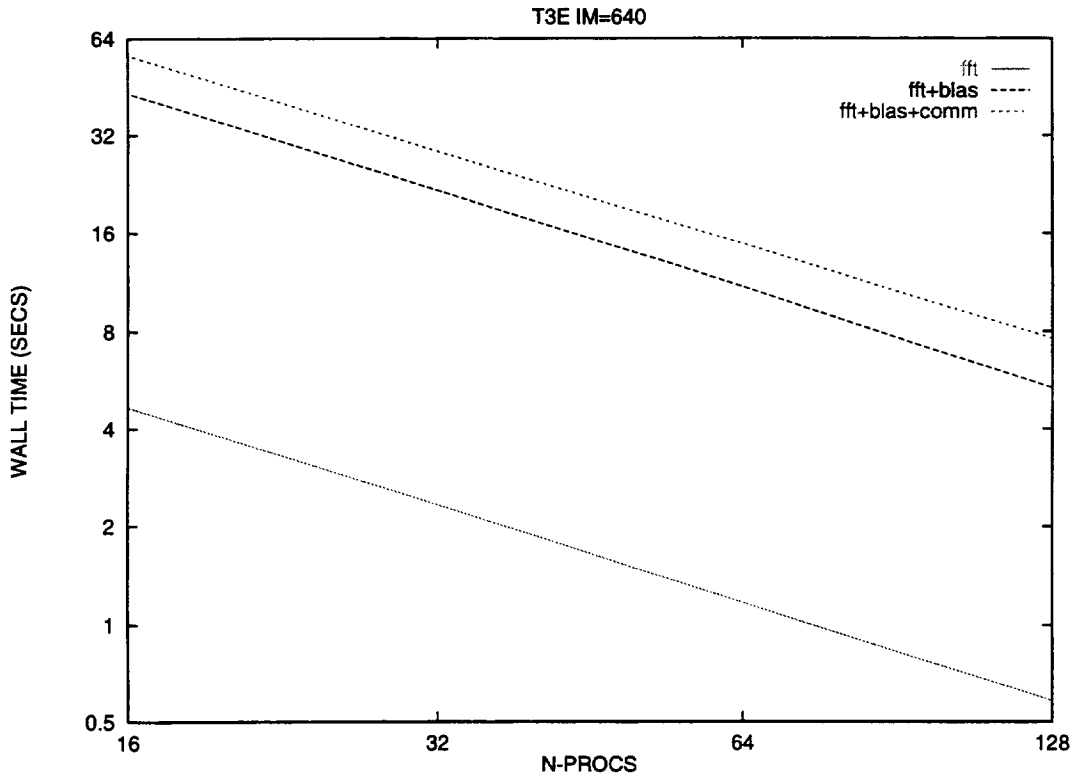


Figure 2: FFT, Legendre transform (BLAS SGEMM), and communication times for T213L32

4. Overall model performance:

Table 1 shows NOGAPS running times for both the T3E and Origin 2000 for a range of processor numbers. The results show relatively poor scaling going from 60 to 120 processors on the T3E and also from 30 to 60 processors on the O2000. This was largely due to poor load balancing, primarily in the cumulus parameterization and longwave radiation, and terrible scaling for the communication, particularly on the O2000. The latter problem is clearly a top priority as we continue development of the new model. The load balance problems are solvable with a more elaborate interleaving of latitude and longitude bands in the Gaussian grid point fields.

Table 1: T3E/O2K T159 NOGAPS performance (48 hr forecast, time in secs)

	T3E/p30	T3E/p60	T3E/p120	O2K/p30	O2K/p60
Total:	3780	2100	1400	4589	3242
Diabat:	1357	790	575	1171	905
Cumulus:	366	196	150	226	125
Lwrad:	100	78	88	167	297
Swrad:	169	86	44	98	48
Comtrans:	633	358	234	1295	1039
Comlwrad:	16	35	64	124	273

Table 2 shows single node performance for a quite small (T21) NOGAPS, chosen to fit on the relatively small T3E on-node memory. The model is essentially the current operational code, highly vectorized for the C90, although at this resolution the average vector lengths are quite short. Perhaps the most conspicuous result is the relatively poor performance of the T3E relative to the DEC ALPHA. In spite of a 50% faster processor speed, the smaller cache of the T3E ALPHA processor cause substantially poorer performance then the DEC.

Table 2: T21L18 NOGAPS single node performance (24 hour fcst, time in secs)

	ALPHA 300 SMP	ALPHA 450 T3E	O2000	Cray C90
Total:	106.4	192.8	115.6	32.0
Trans:	9.7	25.6	12.0	3.6
Diabat:	91.4	162.0	100.1	26.2
Cumulus:	9.4	12.4	13.2	3.9
Lwrad:	27.2	56.4	32.7	6.7
Swrad:	23.4	30.0	22.5	5.4

5. Summary:

We have begun the process of converting a large operational NWP model code, optimized for a parallel vector architecture, to a yet to be determined scalable architecture. The code is as general as possible to ensure reasonably graceful porting to a variety of candidate architectures and programming models. Some inefficiencies are inevitable with this philosophy, but if we understand the reasons for these problems, we believe future refinements of the model will eliminate them. An important point to be made is that no effort has yet been made to redesign the model for more optimum performance on a distributed memory, cache-based microprocessor system. Specifically, the model carries its time level histories in spectral space, rather than Gaussian grid point space. This conserves memory, but generally requires more transform operations per time step. On shared memory vector platforms such as the C90 this is an attractive design strategy, since transforms are relatively cheap, but on distributed memory ar-

chitectures there is a considerable penalty in both computational and communication cost with this approach. The model also preserves a rich complement of in-line global diagnostics which are critical for monitoring meteorological performance but would be an expensive luxury in a scalable production code.

One of the greatest challenges of moving to these scalable architectures is accepting the fact that these systems are not all-purpose, and many of the "whistles and bells" that our models now contain will have to be removed, or at least made optional. This has potentially important impacts on the user-friendliness of many models which are often run by relatively naive users. NOGAPS is such a model, and we cannot ignore the implications on software design and configuration management.

References

[1] Hogan, T. F. and T. E. Rosmond, 1991: The description of the Navy Global Operational Prediction System's spectral forecast model. *Mon. Wea. Rev.* **119**, 1786-1815.

[2] Foster, I. and P. Worley, 1993: Parallelizing the spectral transform method: A comparison of alternative parallel algorithms. Proceedings of the 1994 Scalable High Performance Computing Conference, Knoxville, TN, May 23-25, IEEE, Los Alamitos, Ca. 726-733.

[3] Barros, S. R. M., D. Dent, L. Isaksen, and G. Robinson, 1995: The IFS Model - Overview and parallel strategies. Coming of Age: Proceedings of the Sixth ECMWF Workshop on the Use of Parallel Processors in Meteorology. Eds. Hoffman, G-R., and N. Kreitz. World Scientific, Singapore. 303-318.

Navy Weather and Oceanography in the Next Century - A New Challenge in Numerical Modeling

K. D. Pollak and C. J. Mauck
Fleet Numerical Meteorology and Oceanography Center
7 Grace Hopper Avenue, Stop 1
Monterey, CA 93943
E-mail: kpollak@fnmoc.navy.mil
+1 408 656-4335
Fax: +1 408 656-4489

ABSTRACT

The Fleet Numerical Meteorology and Oceanography Center (FNMOC) in Monterey currently runs a suite of meteorological and oceanographic (METOC) analysis and forecast models on Cray C90 and J90 mainframes. These models, which provide METOC support throughout the Department of Defense, are becoming obsolete as their resolutions and physics packages are limited by current operational computer hardware architecture. The Navy also faces the challenge of implementing more skillful models. Fortunately for the Navy, the science and technology of numerical modeling has kept pace with computer hardware. The chief impediment to instituting improved models is acquiring a larger computational capability in which to run them.

FNMOC is now formalizing plans to replace its aging vector processing architecture with a scaleable system. This new system will provide an affordable, phased approach to meet FNMOC's new computational requirements. Next generation models executing on this replacement system will provide FNMOC's customers with more skillful and longer range METOC forecasts. Improvements will be the result of better physics, higher grid resolutions, and full coupling of air, ocean, wave, and ice models.

INTRODUCTION

Mission

The mission of FNMOC is "to combine innovative technology with the best available science in order to provide the best weather and oceanographic products, data and services to the operating and support forces of the Department of Defense, anywhere, anytime" (FNMOC [1]). FNMOC provides these services continuously in order to increase safety of forces and to optimize the use of platforms, weapons, sensors and facilities.

Keys to its mission are responsiveness and quality. To do this, FNMOC maintains its competitive abilities by applying better physics to models on a regular basis. Better

physics usually means more calculations are required, resulting in longer model run times. Model improvements are also made by decreasing the computational grid spacing, allowing a model to resolve smaller scale features in the ocean or atmosphere. Reducing a model's grid spacing while maintaining the same geographic domain has two affects: it increases the amount of computer memory required, and it increases model run time. The capacity of a given computer architecture thus effectively limits the size and number of models that can be run.

Over the life cycle of FNMOC's current Cray/workstation computer center, there has been a metamorphosis of sorts. As models were implemented and incrementally improved, the host computers were also upgraded with better operating systems, more memory, and additional processors. There is a point though when hardware improvements are either not possible or no longer cost effective. As FNMOC's hardware system becomes saturated and future improvements are no longer economically viable, the entire hardware/software system will become static. Hardware vendors in pursuit of newer technology may no longer be willing to support "older" systems. The current configuration at FNMOC will thus come to the end of its life cycle. To remain competitive, a computing capability with more memory and better technology will be needed.

CAPABILITIES

The product list in Table 1 shows examples of tailored applications that use FNMOC meteorological and oceanographic products. Model fields are also transferred electronically directly to customers for use onboard ships, at regional Navy forecast centers, and other government agencies.

Product	Required Input
Optimum Aircraft Routing Services	Meteorological and Oceanographic Model Products
Optimum Track Ship Routing Services	Meteorological and Oceanographic Model Products
Tropical Storm Predictions and Warnings	Meteorological and Oceanographic Model Products
High Winds and High Seas Warnings	Meteorological and Oceanographic Model Products
Precipitation Products	Meteorological Model Products
Refractivity Conditions and Ducting Ranges	Meteorological Model Products
Underwater Acoustic Support	Oceanographic Model Thermal Structure Products
Support Low Level Atmospheric Release Predictions	Meteorological Model Winds
Search and Rescue	Oceanographic Model Surface Currents, Winds

Table 1. Warfighter Support.

EXISTING HARDWARE AND SOFTWARE CONFIGURATION

HARDWARE CONFIGURATION

FNMOC's current computer hardware system is anchored by a sixteen processor Cray C90 with 256 Mwords of memory. The C90 is complemented by two eight-processor Cray J90s, each with 512 Mwords of memory. One of the J90s is used as the primary relational data base management system server, while the second provides a bridge between the C90 and peripheral systems, runs applications and hosts a capability for running lower resolution METOC models in a backup mode. Seventy-one Sun SparcStations round out FNMOC's computing environment. These workstations perform a myriad of functions including: communications, data base servers, applications, and development. These Sun computer systems range from 12-processor, 1024 Mbyte SPARCCenter 2000Es, to SPARC 10 desktop workstations. Figure 1 depicts the FNMOC computer architecture.

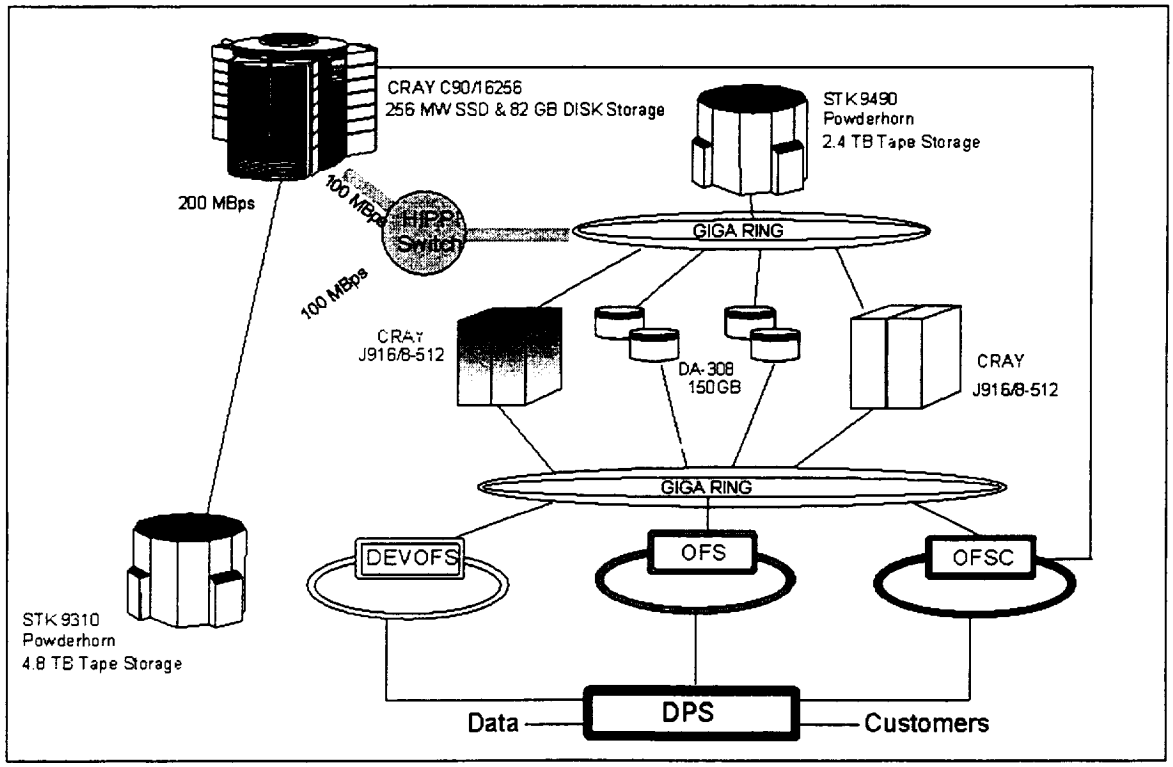


Figure 1. FNMOC Computer Architecture. DEVOFS is the developmental file server, OFS is the unclassified operational file server, and OFSC is the classified operational file server. All three are SparcStations. DPS, the distributed processing system, is a suite of workstations which provide the primary communication interface to customers.

MODEL SOFTWARE DESCRIPTION

The major models now operational at Fleet Numerical are briefly described here.

Meteorological Models

NOGAPS - The Navy Operational Global Atmospheric Prediction System (NOGAPS) model is a global spectral numerical weather prediction model (Hogan and Rosmond [2]). NOGAPS employs state-of-the-art data quality control, data assimilation, nonlinear normal mode initialization, and atmospheric physics to produce skillful medium-range weather forecasts. NOGAPS generates several thousand operational fields per day, including surface winds and heat fluxes to drive ocean models and lateral boundary conditions to support regional atmospheric models. In one way or another, NOGAPS output supports nearly every operational application run at Fleet Numerical. It is the only global meteorological model operated by DoD. In its 159 spectral wave configuration with 18 levels, NOGAPS makes operational forecasts to 144 hours twice daily. The model then runs in a 63 spectral wave configuration to 240 hours. This configuration uses approximately 80 million words of memory and requires around 12 minutes of wall time per forecast day, though this requirement depends on the time step being used and on competition for resources.

NORAPS - The Navy Operational Regional Atmospheric Prediction System (NORAPS) model is a relocatable regional primitive equation numerical weather prediction model (Hodur [3]). NORAPS is run at higher horizontal and vertical resolution than NOGAPS for areas of high DoD interest. It can be initialized either from its own high-resolution nowcast, or from the coarser resolution NOGAPS nowcast. It uses lateral boundary conditions provided by NOGAPS, and generally provides a more accurate and detailed depiction of mesoscale weather features than NOGAPS, particularly in areas affected by the land surface. NORAPS currently runs in one of three resolutions, 45 km, 20 km, and a 15 km resolution nest within a 45 km resolution area. Currently, FNMOC runs six NORAPS areas operationally (2-45/15 km, 2-45 km, 2-20 km). The 45 km resolution areas use around 40 million words of memory and require 35 minutes to complete a 48 hour forecast.

COAMPS - NORAPS is being replaced with the Coupled Ocean/Atmosphere Mesoscale Prediction System (COAMPS) model (Hodur [4]). The atmospheric component of COAMPS features triply-nested grids down to resolutions of a few kilometers, non-hydrostatic dynamics, explicit moisture physics and aerosols, and improved data assimilation. The underlying and fully coupled oceanographic component of COAMPS will eventually combine the functions of OTIS (ocean thermal data assimilation), POM (ocean thermal and circulation prediction) and WAM (ocean wave prediction) to provide for fully interactive two-way coupling between ocean and atmosphere (Clancy and Hodur [5]). With lateral boundary conditions provided by FNMOC global models, COAMPS provides the high-resolution, relocatable and fully integrated METOC prediction capability required for seamless support of the sea-air-land operations implied by the

Navy's new missions. Currently, FNMOC runs two COAMPS areas, Europe and Southwest Asia. The model for each area is characterized by an 81 km outer grid, and inner grids of 27 km and 9 km. The Southwest Asia (larger area) uses around 100 million words of memory and requires approximately 2.5 hours to complete a 48 hour forecast. FNMOC runs each COAMPS area twice daily.

EFS - The Ensemble Forecast System (EFS) is implemented with a coarse horizontal resolution version of NOGAPS (Pauley *et al.* [6]). In this state-of-the-art approach, multiple forecast runs are made from slightly differing initial conditions, with each obtained by means of a process that "breeds" the growing error modes that dominate forecast error (Toth and Kalnay [7]). By averaging the resulting multiple forecast realizations (and hence tending to cancel out the effect of the growing error modes), a forecast is achieved with higher skill than any single forecast produced even with a higher resolution version of the model. In addition, the spread of forecast realizations allows an estimate to be made of the range of forecast error, which can vary substantially from week to week depending on the global-scale flow patterns in the atmosphere.

GFDN Tropical Cyclone Model - The Geophysical Fluid Dynamics Laboratory (GFDL) Tropical Cyclone Model is implemented at Fleet Numerical to provide track and intensity predictions for hurricanes and typhoons. The model is described by Kurihara *et al.* [8], and includes a moving triply-nested grid, second order turbulence closure, convective adjustment, infrared and solar radiation, and parameterization of land surface characteristics by vegetation type. The model is initialized from a special analysis constructed by removing the tropical cyclone component from the NOGAPS analysis and replacing it with a synthetic vortex generated from the observed location and structure of the storm. Forecast lateral boundary conditions for the Tropical Cyclone Model forecasts are provided by NOGAPS.

Oceanographic models

WAM - The Third-Generation Wave Model (WAM) contains state-of-the-art nonlinear physics for forecasting the evolution of directional wave energy spectra and derived wave height, period and direction fields (WAMDI Group [9]). WAM is run in both global coarse-resolution and regional high-resolution implementations at Fleet Numerical. The regional implementations generally include shallow water physics to account for refraction and bottom friction effects, although these formulations begin to lose validity at depths shallower than about 30 meters. WAM uses wind stress forcing provided by either NOGAPS, NORAPS or COAMPS. WAM provides crucial support for Optimum Track Ship Routing (OTSR), the issuance of high-seas warnings, and many other applications. There are currently one global and four regional WAM areas that run on the c90, and five areas running on a j90. Global WAM, the largest of the all areas, requires 70 million words of memory and takes about 45 minutes of wall time for a 5 day forecast.

OTIS - The Optimum Thermal Interpolation System (OTIS) is the primary ocean thermal nowcast model used at Fleet Numerical (Cummings [10]). Both global coarse-resolution

and regional high-resolution versions are in use. All of the OTIS implementations use the Optimum Interpolation (OI) technique to assimilate real-time data. Regional OTIS further employs water-mass-based representation of ocean thermal climatology and ocean front and eddy "feature models" to produce "synthetic" data to supplement the "real" data. This allows a detailed and accurate depiction of subsurface thermal structure associated with fronts and eddies whose surface positions are depicted in operational ocean front and eddy analyses derived primarily from satellite imagery by analysts at the Naval Oceanographic Office. OTIS runs globally for surface only analysis, and separately for a full surface to 5000 meters (3-D) analysis. There are three regional areas running 3-D analyses: western Pacific, western Atlantic, and Greenland/Norwegian Seas. Separate sea surface temperature OTIS analyses also run for each NORAPS area. OTIS uses about 42 million words of memory and takes approximately 30 minutes of wall time for its largest area, a 28 Km global analysis of sea surface temperature.

TOPS - The Thermodynamic Ocean Prediction System (TOPS) is a synoptic ocean mixed-layer model (Clancy and Pollak [11]). Both global coarse-resolution and regional high-resolution versions are in use. TOPS is initialized by temperature and salinity fields nowcast by OTIS, and includes sophisticated turbulence closure physics and radiation absorption calculations. TOPS produces forecasts of upper-ocean thermal structure and currents driven by surface wind stresses and heat fluxes predicted by either NOGAPS or NORAPS. Three regional areas, one for each 3-D OTIS area, and a global TOPS, each run once per day. TOPS uses approximately 13 million words of memory and requires about 20 minutes of wall time to complete a 72-hour forecast for the global 110 Km configuration.

PIPS - The Polar Ice Prediction System (PIPS) is a dynamic and thermodynamic sea-ice model designed to forecast ice thickness, concentration and drift in the arctic (Cheng and Preller [12]). PIPS is driven by surface wind stresses and heat fluxes from NOGAPS, and is coupled with an underlying dynamic ocean model. PIPS is updated daily from an objective analysis of ice concentration data from the Special Sensor Microwave/Imager (SSM/I) instrument aboard the Defense Meteorological Satellite Program (DMSP) satellites. PIPS uses 41 million words of memory and takes about 45 minutes of wall time to run.

DART - The Data Assimilation Research Transition (DART) model is a two-layer primitive equation dynamic ocean model designed to forecast the evolution of the Gulf Stream (Thompson and Schmitz [13]). It currently produces two-week forecasts of Gulf Stream north-wall positions.

The following models are currently being implemented and tested for possible operational use at FNMOC.

POM - The Princeton Ocean Model (POM; Blumberg and Mellor [14]) is a multi-level primitive equation ocean circulation model, which contains a sophisticated treatment of vertical mixing. The model includes atmospheric and tidal forcing and is designed

specifically for high-resolution shallow-water applications in support of the Navy's new emphasis on coastal operations. Initial testing of POM at Fleet Numerical has been for the West Coast of the United States (Clancy *et al.* [15]) and for the Yellow Sea (Riedlinger and Preller [16]). POM has already been used operationally by the Navy in semi-enclosed seas where lateral open boundary conditions are not an issue (Horton, *et al.* [17]). In general, POM is expected to be the Navy's model-of-choice over the next several years for providing high-resolution coastal predictions of currents, sea level and thermal structure.

NLOM - The Navy Layered Ocean Model (NLOM) is a global and, at least, marginally eddy resolving implementation of the Navy Layered Ocean Model of Wallcraft [18], which is a descendant of the model of Hurlburt and Thompson [19]. NLOM will support coastal implementations of POM through lateral boundary conditions, and provide an improved representation of ocean currents on the global scale. Assimilation of satellite altimeter data into NLOM and POM will be a crucial requirement for their success.

MOM – The GFDL Modular Ocean Model (MOM) (Pacanowski [20]) is a three-dimensional primitive equation ocean model. . MOM was designed to run on large-scale vector processors such as FNMOC's Cray C90. It has been used with NOGAPS in coupled air/ocean research at the Naval Research Lab in Monterey (Li and Hogan [21]). FNMOC is currently using version 2 (MOM 2). It contains state-of-the-art physical parameterizations with an implicit free surface and surface mixing. MOM 2 is being tested on a global one-half degree latitude and longitude grid, with 20 vertical levels. With appropriate data assimilation, MOM 2 may provide improvements over global 3-D OTIS and TOPS. Expected products from MOM 2 would be surface and subsurface total currents (TOPS provides only wind mixed surface currents), temperature, salinity, dynamic height and mixed layer depth.

THE CHALLENGE

Within the next three years, FNMOC must complete a transition of all its hardware and software to a more capable system. Although the hardware vendors for the new system are not yet identified, the specifications are now being determined for targeted software requirements. With this planned increased computer power, numerical prediction models will have fewer constraints on computer memory and run time limitations. For this transition to take place, existing meteorological and oceanographic models will either have to migrate to the new scalable computer architecture with appropriate reprogramming, or be replaced with new software. The ultimate challenge will be to combine relatively new hardware and software technology into a capable, operational system by the year 2001.

Table 2a summarizes expected capabilities for the future generation of meteorological models at FNMOC. Hardware specifications for FNMOC's scalable architecture will allow the global weather model to execute over its entire forecast length of ten days, twice per day and taking five hours of wall clock time per ten day forecast. Specifications

also call for the capability to run at least five regional area models. Each regional model will be able to nest smaller, higher resolution models on mesoscale, tactical, and battlefield domains. Nesting is a means of simultaneously executing the same regional model for different scaled domains provided they are contained within each other. Table 2a also shows the number of model levels, fields, and storage requirements for the global model and five nested regional models.

	GLOBAL	REGIONAL	MESOSCALE	TACTICAL	BATTLEFIELD
Domain Size	Global	9000 X 9000 km	3000 X 3000 km	1000 X 1000 km	333 X 333 km
Grid Resolution	50 km (~.5 degrees lat/lon.)	54 km	18 km	6 km	2 km
Vertical Levels	50 Sigma 30 Pressure	50 Sigma 30 Pressure	50 Sigma 30 Pressure	50 Sigma 30 Pressure	50 Sigma 30 Pressure
Multi Level Parameters	6	6	6	6	6
Single Level Parameters	30+	30+	30+	30+	30+
Forecast Periods	81 (240 hours in 3 hour increments)	17 (48 hours in 3 hour increments)	17 (48 hours in 3 hour increments)	9 (24 hours in 3 hour increments)	25 (24 hours in 1 hour increments)
2-D Press Level Fields	17010	3570	3570	1890	5250
2-D Sigma Level Fields	24300	5100	5100	2700	7500
Grid Points per 2-D Field	259920	27789	27789	27789	27789
Application Grids		3570	3570	1890	5250
IEEE Storage	~43 Gbytes	~7 Gbytes	~7 Gbytes	~4 Gbyte	~10 Gbytes

Table 2a. Future meteorological model requirements. The Battlefield scale is a required capability, but may not be invoked in all areas.

One of the strengths of the new model software designs being developed at FNMOC is the ability to couple the meteorological models with oceanographic models. Coupling could help account for positive feedback and air-sea interactions (Li [22,23]). COAMPS will become FNMOC's regional coupled air-sea model as ocean submodel components are added progressively to it. FNMOC will look toward the "Ocean" part of COAMPS to meet many of the customer requirements for oceanographic products.

NOGAPS will be reconfigured to run on FNMOC's new scalable architecture and will continue to provide global atmospheric products. The concept of coupling a global ocean circulation model with NOGAPS, in some fashion, is also being studied. There are at least two ocean circulation model candidates that could be selected for this purpose. One is being developed at the Naval Research Laboratory at Stennis Space Center in Mississippi. It is the NRL Coastal Ocean Model (NCOM). NCOM is expected to be the future ocean component for COAMPS, and may also be delivered to run in a global

configuration. If a global version becomes available, it could run with either one-way interaction with NOGAPS (forced with NOGAPS winds and heat fluxes), or run in a more closely coupled mode. The second choice is a descendent of MOM, called the Parallel Ocean Program (POP; Semtner [24]). POP was reprogrammed at the Los Alamos National Laboratory for use on parallel computers with distributed memory. Due to the common heritage between MOM and POP, the transition of MOM from the Cray computers to POP on a parallel architecture makes POP a strong candidate for one or two-way coupling with NOGAPS.

Future ocean wave models will also need to be reprogrammed to run on the scalable computer architecture. The migration of WAM is currently being funded in part by the Common High Performance Computing Software Support Initiative (CHSSI). This is a component of the Department of Defense High Performance Modernization Program (West, *et al.* [25]). Migration of TOPS, OTIS and PIPS is still an issue. Replacement rather than migration may be the answer for TOPS and OTIS. Here, with appropriate data assimilation, NCOM could prove to show more skill. The danger, however, of seeking replacements instead of migrating the existing models is that FNMOC would be forced to rely on new and unproven technology to replace proven operational products. Table 2b summarizes the desired capabilities of future oceanographic models.

	GLOBAL	MESOSCALE	TACTICAL	BATTLEFIELD
Domain Size	Global	3000 X 3000 km	1000 X 1000 km	333 X 333 km
Grid Resolution	25 km (~.25 degrees lat/lon.)	18 km	6 km	2 km
Vertical Levels	35	35	35	35
Multi-level Parameters	7	7	7	7
Single-level Parameters	20+	20+	20+	20+
Forecast Periods	41 (240 hours in 6 hour increments)	17 (48 hours in 3 hour increments)	13 (36 hours in 3 hour increments)	25 (24 hours in 1 hour increments)
2D Fields	10865	4505	2385	6625
Grid Points per 2D Field	1038240	27889	27889	27889
IEEE Storage	~46 Gbytes	~3 Gbytes	~2Gbytes`	~4Gbytes

Table 2b. Future oceanographic model requirements. The Tactical and Battlefield scales are required capabilities, but may not be invoked in all areas.

CONCLUSION

Confronted with an aging computer system that has become nearly saturated with operational applications, FNMOC is faced with a new challenge over the next three years. This challenge will be to replace its existing system with an affordable, yet capable, new system. The transition will target a scalable system. FNMOC will take advantage of new technology both in hardware improvements and in numerical modeling advancements available from the scientific community. Since vendor support for the existing system is

becoming costly with very little end benefit, the new hardware/software configuration must be in place within three years.

REFERENCES

1. Fleet Numerical Meteorology and Oceanography Center, 1987: Strategic Plan, 7 Grace Hopper Avenue, Stop 1, Monterey, CA 93943

2. Hogan, T.F., and T.E. Rosmond, 1991: The description of the Navy Operational Global Atmospheric System's spectral forecast model. *Monthly Weather Review, 119,* 1786-1815.

3. Hodur, R.M., 1987: Evaluation of a regional model with an update cycle. *Monthly Weather Review, 115,* 2707-2718.

4. R. M. Hodur, 1997. The Naval Research Laboratory's Coupled Ocean/Atmosphere Mesoscale Prediction System (COAMPS). *Monthly Weather Review, 125,* 1997, pp. 1414-1430.

5. Clancy, R.M. and R. M. Hodur, 1996. Projected five-year evolution of the Coupled Ocean/Atmosphere Mesoscale Prediction System (COAMPS). *Proceedings of the American Meteorological Society Conference on Coastal Oceanic and Atmospheric Prediction*, Atlanta, GA, 28 January through 2 February 1996, pp. 68-71.

6. Pauley, R., M.A. Rennick, and S. Swadley, 1996: Ensemble forecast product development at Fleet Numerical Meteorology and Oceanography Center. Tech Note, Models Department, FNMOC, Monterey, CA 93943-5501.

7. Toth, Z. and E. Kalnay, 1993: Ensemble forecasting at NMC: the generation of perturbations. *Bulletin of the American Meteorological Society, 74,* 2317-2330.

8. Kurihara, Y., M.A. Bender, R.E. Tuleya, and R.J. Ross, 1995: Improvements in the GFDL Hurricane Prediction System. *Monthly Weather Review, 118*, 2186-2198.

9. WAMDI Group, 1988: The WAM model - A third-generation ocean wave prediction model. *Journal of Physical Oceanography, 18,* 1775-1810.

10. Cummings, J.A., 1994: Global and regional ocean thermal analysis systems at Fleet Numerical Meteorology and Oceanography Center. Proceedings of the MTS'94 Conference, 7-9 September 1994, Washington, DC, Marine Technology Society, 1828 L Street NW, Suite 906, Washington, DC 20036.

11. Clancy, R.M., and K.D. Pollak, 1983: A real-time synoptic ocean thermal analysis/forecast system. *Progress in Oceanography, 12,* 383-424.

12. Cheng, A., and R.H. Preller, 1992: An ice-ocean coupled model For The Northern Hemisphere. *Geophysical Research Letters, 19,* 901-904.

13. Thompson, J.D., and W.J. Schmitz, 1989: A limited-area model of the Gulf Stream: Design and initial experiments. *Journal of Physical Oceanography, 19,* 791-814.

14. Blumberg, A.F., and G.L. Mellor, 1987: A description of a three-dimensional coastal circulation model. In: Three-Dimensional Coastal Circulation Models, N. Heaps, ed., American Geophysical Union, Washington, DC, 208 pp.

15. Clancy, R.M., P.W. deWitt, P. May and D.S. Ko, 1996: Implementation of a coastal ocean circulation model for the west coast of the United States. Proceedings of the American Meteorological Society Conference on Coastal Oceanic and Atmospheric Prediction, Atlanta, GA, 28 January through 2 February 1996, pp. 72-75.

16. Riedlinger, S. and R. Preller, 1996. The Development of a Dynamic/Thermodynamic Numerical Model of the Yellow Sea/East China Sea Region, Proceeding of the American Meteorological Society's Conference on Coastal Oceanic and Atmospheric Prediction, Atlanta, pp52-79.

17. Horton, C., M. Clifford, J. Schmitz and B. Hester, 1994: SWAFS: Shallow Water Analysis and Forecast System overview and status report. Technical Report, Naval Oceanographic Office, Stennis Space Center, MS 39522, 53 pp.

18. Wallcraft, A.J., 1991: The Navy layered ocean model users guide. NOARL Report 35, Naval Research Laboratory, Stennis Space Center, MS 39529.

19. Hurlburt, H.E. and J.D. Thompson, 1980: A numerical study of loop current intrusions and eddy shedding. *Journal of Physical Oceanography, 10,* 1611-1651.

20. Pacanowski, R.C., 1995: MOM 2 Documentation User's Guide and Reference Manual, Version 1.0, GFDL Ocean Technical Report #3.

21. Li, T. and Hogan T.F. 1998: The Role of the Annual-Mean Climate on Seasonal and Interannual Variability of the Tropical Pacific in a Coupled GCM. Submitted to Journal of Climate.

22. Li, T. 1997: Air-Sea Interactions of Relevance to the ITZC: Analysis of Coupled Instabilities and Experiments in a Hybrid Coupled GCM. Journal of Atmospheric Sciences, 54,134-147.

23. Li, T. 1997: Phase Transition of the El Nino-Southern Oscillation: A stationary SST Mode. Journal of Atmospheric Sciences, 54,2872-2887.

24. Semtner, A.J.,1997: Introduction to "A Numerical Method for the Study of the Circulation of the World Ocean". Journal of Computational Physics, 135, 149-153.

25. West, J.E., Jensen, R.E., and Turcotte, L.H., 1997: Migration of WAM to Scalable Computing Environments, U.S. Army Corps of Engineers, Waterways Experiment Station, Technical Report ITL-97.

Parallelization of a spectral atmospheric GCM

V. Balaji*

NOAA/GFDL, P.O Box 308

Princeton University

Princeton NJ 08542

May 20, 1998

Abstract

The spectral transform has proved a robust method for the treatment of the non-linear adiabatic Navier-Stokes equations of fluid flow on a sphere. It is robust and the intrinsic shortcomings of spectral methods (dispersion and truncation, and the formation of Gibbs ripples in the presence of steep gradients) are well understood. It is therefore one of the preferred forms for global atmospheric models. In the spectral transform method each field has a representation in spectral coefficients of spherical harmonics, and a corresponding grid field. The linear dynamics is generally treated in spectral space, while the physics (i.e gridscale parameterization of sub-gridscale processes and thermodynamics) and the non-linear terms are treated in grid space.

*Senior Applications Analyst, Silicon Graphics/Cray Research.

A new spectral dynamical core is being developed at GFDL. This model is entirely coded in f90 and takes a modular, object-oriented approach. Each field is represented by a grid object and a spectral object with a standard interface. The transform module can convert one to the other. Separate modules exist for operations that are done entirely in spectral space and grid space. All the physics modules treat the grid objects, and since the interface is standard, the physics in this model is entirely modular.

The current paper concerns the parallelization of this spectral dynamical core. The approach that has been taken remains entirely consistent with the modular object-oriented approach. A flexible, dynamic data decomposition module has been written which takes the grid and spectral objects and manages the distribution and communication of data among independent processors. The decomposition is entirely dynamic and run-time configurable, and divides grid or spectral space into data and computational domains. The domains can be global or local at various stages of the transform depending on our tolerance for the communication overhead. This permits various strategies to minimize inter-processor communication in the two stages of the spectral transform. We demonstrate different methods of implementing this within the dynamic decomposition approach. Finally, we also present performance of the parallel spectral dynamical core on a Held-Suarez climate benchmark.

1 Introduction

All numerical models for the atmospheric general circulation (GCMs) solve discrete forms of the non-linear Navier-Stokes equations for fluid flow in a spherical shell, coupled with equations describing the thermodynamic state of air. The prevalent terminology divides

these equations into the *dynamics* and *physics*, where the former refers to the resolved Navier-Stokes dynamics, with the parameterization of thermodynamics and unresolved scales being termed the physics. The division can be somewhat arbitrary, and is obviously a function of the resolution scale. For climate problems in particular, it is necessary to develop numerical representations of these equations that not only preserve long-term statistical integrity, but are also computationally efficient enough for the very long integrations that these studies require.

The design of the codes in which these simulations are carried out remains in a dialectical relationship with the evolution of computer architectures and compiler technologies. In particular, codes are in transition now from an era dominated by vector supercomputers to one where the Trades blow in the direction of massively parallel processing architectures. Furthermore, as interest grows in simulation of the effects of climate change at regional and smaller scales, there is a trend toward increasing resolution, and the phenomena represented in the physics packages of models remains in constant evolution. In fact, given the wide variety of atmospheric GCM code designs currently in existence, a benchmark has been proposed for the intercomparison of their performance and integrity over climate timescales (Held and Suarez 1994). It is not a conventional benchmark in that what will be compared is the equilibrium statistical behaviour (the "climate") of the models rather than their convergence toward an exact solution.

The twin needs of scalability and modularity have prompted the design of a new generation of models at GFDL. These models are being designed with a maximum of flexibility in mind. Not only may these codes have to perform over a range of computing architectures, they will also be in constant flux, especially in regard to the physics packages with which

they are used. With this in mind, we have isolated the *dynamical core* of these models as far as possible. The dynamical core accepts a single uniform module interface for physics. Thus, the physics packages remain entirely interchangeable. The dynamical cores themselves also retain a modular texture, as a wide variety of algorithms may be used for various aspects of the dynamics depending on the problem being solved and the available computing power on the platform upon which the calculations are being carried out.

The new generation models include a finite-difference gridpoint model and a spectral dynamical model. This paper is concerned with the modular design of a scalable dynamical core for the spectral model.

In Sec. 2 we describe the spectral transform method that is generally used for spectral dynamical cores. In Sec. 3 we analyze the data dependency patterns of the spectral transform method and construct an algorithm for the spectral transform method in Fortran 90 that permits us to evaluate a range of possible parallel implementations.

2 The spectral transform method.

A spectral representation consists in constructing the dynamical fields as expansions in spectral basis functions. The expansions are truncated at some chosen order for numerical purposes. Spherical harmonics are generally chosen as spectral basis functions. The problem arises with non-linear terms, such as in advection, which, if explicitly expanded, result in a sum of terms quadratic in the order of the expansion. The transform method (Orszag 1970) consists in returning the fields to grid space for the computation of non-linear terms, and transforming back for the spectral computations. Since this method involves frequent

transformations between grid fields and spectral fields, efficient numerical transform methods have been developed, including parallel methods (Foster et al. 1992).

The expansion of a quantity $f(\theta, \phi)$ in spherical harmonics is as follows:

$$f(\theta, \phi) = \sum_{m=-\infty}^{\infty} \sum_{n=|m|}^{\infty} f_{mn} P_{mn}(\cos\theta) e^{im\phi} \tag{1}$$

where (θ, ϕ) are the spherical co-ordinates, m and n are termed the *Fourier wavenumber* and the *spherical wavenumber* respectively, and P_{mn} are the associated Legendre polynomials of the first kind of order m and degree n. The spherical harmonics $Y_{mn} \equiv P_{mn} e^{im\phi}$ satisfy the orthogonality relationship:

$$\frac{1}{4\pi} \int_{-\pi}^{\pi} \int_{\mathrm{SP}}^{\mathrm{NP}} Y_{mn} Y^*_{m'n'} d(\cos\theta)\, d\phi = \delta_{mm'} \delta_{nn'} \tag{2}$$

where the θ integral runs from the South to the North Pole.

The spherical harmonic coefficients f_{mn} of the expansion are given by the inverse transform:

$$f_{mn} = \frac{1}{4\pi} \int_{-\pi}^{\pi} \int_{\mathrm{SP}}^{\mathrm{NP}} f(\theta, \phi) P^*_{mn}(\cos\theta) e^{im\phi} d(\cos\theta)\, d\phi \tag{3}$$

In the numerical representation, these infinite series are evidently truncated. The Fourier wavenumber m is first truncated to a maximum of M. In the *triangular truncation*, the spherical wavenumber n is truncated at $N = M$. In the *rhomboidal truncation*, n is truncated at $|m| + M$. In both cases, M is sufficient to specify N. The truncation is thus specified either as T or R (for triangular or rhomboidal) followed by the value of M, which is sufficient to specify the spherical truncation as well. Thus T42 refers to a spectral expansion that is

triangular-truncated at $M = 42$ and $N = 43$. Henceforth, the summations in Eq. 1 will be understood be truncated at some specified value.

The numerical representation of the integrals in Eq. 3 require a discretization in space of θ and ϕ. The I longitude points ϕ_i are generally chosen to be evenly distributed between 0 and 2π. This facilitates efficient methods for the evaluation of the Fourier integral such as the FFT. For the evaluation of the Legendre integral, the J latitude points θ_j are chosen to lie at the *Gaussian quadrature points.* Note that the Gaussian grid is non-uniform in j. The spectral transform (Orszag 1970) requires $I \geq 3M + 1$ and $J = I/2$.

The numerical form of the transformations between grid space and spectral space is thus as follows:

$$f_{ij} \equiv f(\theta_j, \phi_i) = \sum_{m=-M}^{M} \sum_{n} f_{mn} P_{mn}(\cos\theta_j) e^{-im\phi_i} \tag{4}$$

$$f_{mn} = \frac{1}{4\pi} \sum_{j=1}^{J} \sum_{i=1}^{I} f_{ij} e^{im\phi_i} P_{mn}(\cos\theta_j) \Delta(\cos\theta_j) \Delta\phi \tag{5}$$

$$\tag{6}$$

where the limits of $\sum_n$ depend on the method of truncation, and $\Delta\phi$ is the assumed constant longitude spacing.

The sums are performed in the order shown in Eq. 5 and Eq. 6. The intermediate step produces the partially transformed quantity f_{mj}, which we term the *Fourier representation*:

$$f_{mj} = \sum_{n} f_{mn} P_{mn}(\cos\theta_j) = \sum_{i=1}^{I} f_{ij} e^{im\phi_i} \tag{7}$$

3 Parallelism in the spectral transform.

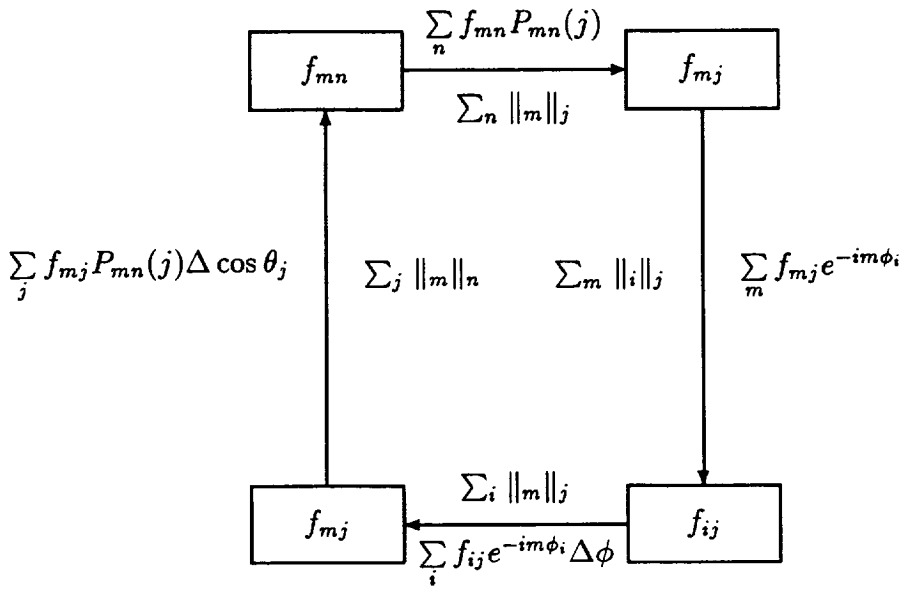
Consider now a model using the spectral transform method, where at each timestep, a field f will have a certain number of operations performed on its spectral form f_{mn} and others on its grid form f_{ij}. Starting with the spectral form f_{mn} we first perform a summation in n, where the summation has to be performed independently over all the possible values of m and j. This step is thus coupled in n and *data-parallel* in m and j, and produces f_{mj}. We then perform a Fourier transform upon f_{mj}, a step that is coupled in m, but data-parallel in i and j. We may now perform any operations upon f that are required to be performed in grid space. Similarly in the reverse transform Eq. 6, we first do an inverse Fourier transform upon f_{ij} that produces a series of independent m coefficients at each j; and finally a reverse Legendre transform that consists of a sum on the Gaussian quadrature points j to reproduce the spectral field f_{mn}. The data dependencies described here are graphically shown in Fig. 1.

Each of these stages involves 3 of the 4 co-ordinate axes m, n, j, i, where a global reduction is performed on one of the axes, and is data-parallel in the other 2. We try to exploit this symmetry in our developmemt of a parallel implementation.[1]

Distributed global reductions are generally discouraged in developing parallel algorithms, since they tend not to scale well with increasing processor counts. While keeping this in mind in developing our parallel implementation, we will evaluate the performance of distributed reductions as well. But for the moment consider the first step of the cycle in Fig. 1, $f_{mn} \rightarrow f_{mj}$. In a parallel implementation, each of the f_{mj}s could in principle be calculated on a

[1]There is the possibility of further parallelism in that there are many such fields f, typically the number of global model variables $\times$ the number of vertical levels per global variable, but in all cases the coupling among fields is very tight, and inhibits parallelism. We will not be considering this sort of parallelism.

Figure 1: Data dependencies in the four steps of the spectral transform.



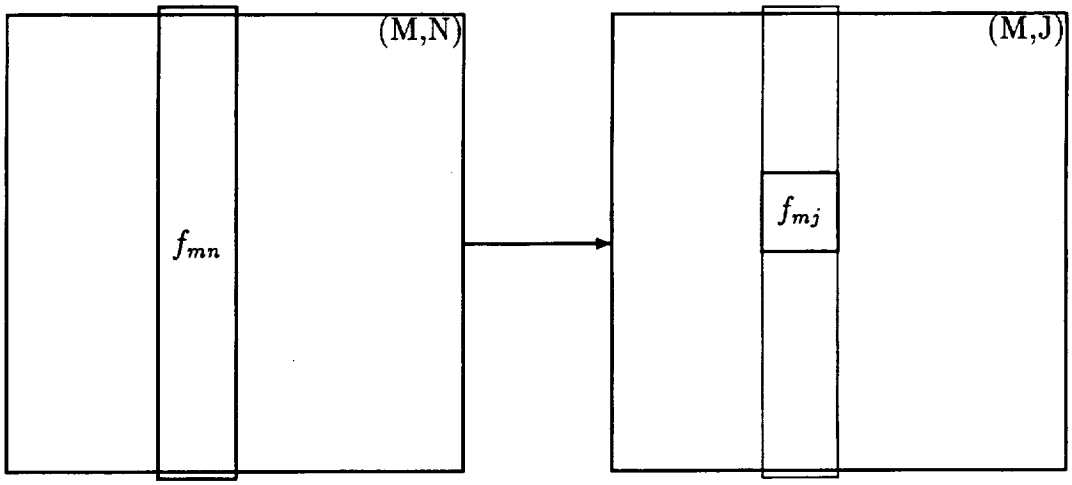
separate processor. But the processors computing f_{mj} for any value of j would *all* require all the values f_{mn} for all values of n, either by performing a global reduction or by acquiring a local copy. This is schematically illustrated in Fig. 2.

Since the entire column f_{mn} in Fig. 2 is required for the computation, this step in the algorithm will be exactly parallel over a maximum of M columns. When the processor count P exceeds M, we expect the scalability to begin falling off. If a large number of processors $P > M$ are available, we may parallelize the computation further:

$$f_{mj} = \sum_{D_n} \left(\sum_{n=\mathrm{NS}}^{\mathrm{NE}} f_{mn} P_{mn}(j) \right) \tag{8}$$

where $n = \mathrm{NS} \cdots \mathrm{NE}$ is the range of n available on the processor that is computing f_{mj}. The sum over the domains D_n is then a global reduction across the processors computing

Figure 2: Data domains in the $f_{mn} \rightarrow f_{mj}$ computation.



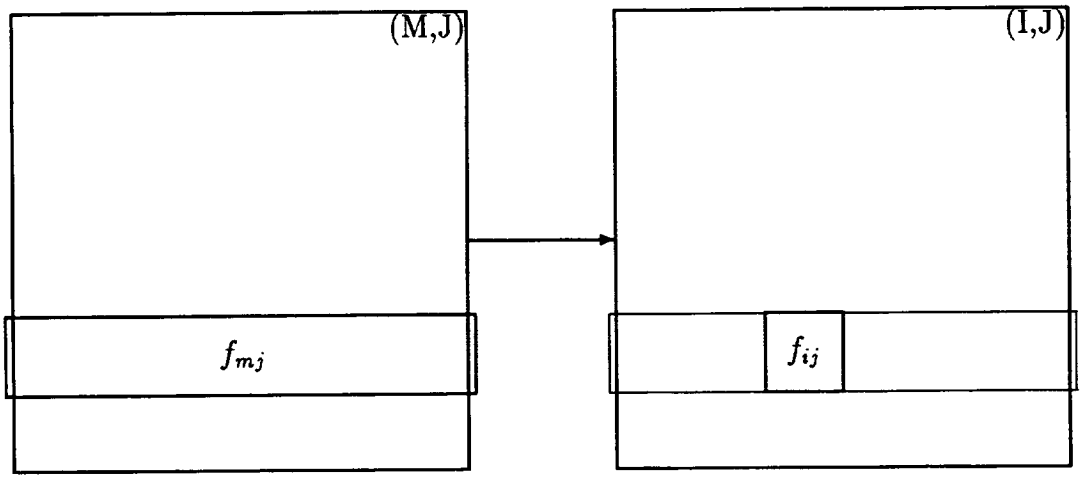
the m column.

Similar considerations hold for each step in the process. For the $f_{mj} \rightarrow f_{ij}$ we have Fig. 3, where now the scalability is limited by J, and all the M values f_{mj} at each j are required for each f_{ij} computation. Similarly, in the reverse order, we have the data dependencies in Fig. 4 for the computation $f_{ij} \rightarrow f_{mj}$, with a scalability limit of I processors, and for $f_{mj} \rightarrow f_{mn}$, we have the data dependencies in Fig. 5, with a scalability limit of N.

We now develop a data distribution scheme based upon these considerations. In its most general form, the decomposition is 2D. We distinguish three spaces: the spectral space S and grid space G, holding the representations f_{mn} and f_{ij} respectively; and the Fourier space F, holding the intermediate form f_{mj}. In each space, data location in memory is specified in terms of a *data domain* and a *computational domain.*[2] The data domain consists of all the

[2]This distinction exists in gridpoint models as well, where the data domain may include a halo region

Figure 3: Data domains in the $f_{mj} \rightarrow f_{ij}$ computation.



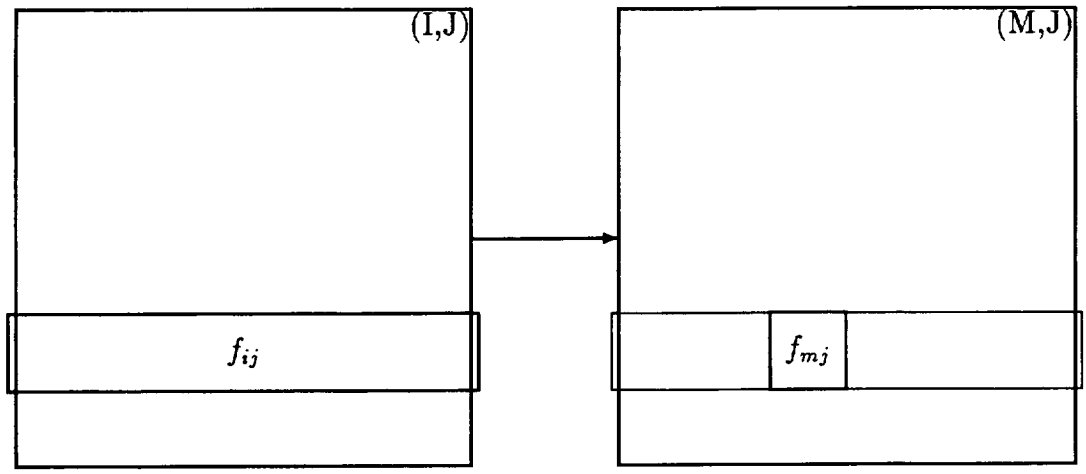
points in some space that is available on a processor, and the computation domain is the set of points which are actually computed. The computational domain in a space is denoted by its name, for the data domain we add a superscript ^{D}. Each domain is considered to be *global* along an axis if it contains all the possible values along that axis, or *local* otherwise. An axis along which a domain is global is given as a subscript. Thus, if the d^{th} domain in spectral space is global in M, we denote this as $S^D_{d;M}$. The data domain is at least as large as the computational domain:

$$H_d \subseteq H_d^D \tag{9}$$

where $H \in \{S, F, G\}$. If the data domain exceeds the computation domain, $H_d \subset H_d^D$, the

around the computational domain.

Figure 4: Data domains in the $f_{ij} \rightarrow f_{mj}$ computation.



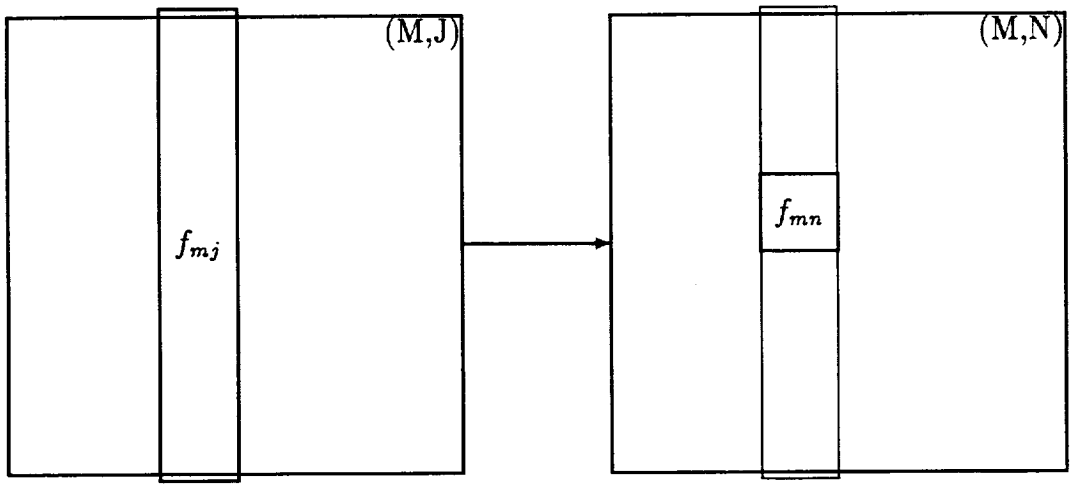
data outside the computational zone must be fetched from the processor on which they are computed. Every data point is computed on at least one processor, but may be redundantly computed on more than one.

$$\bigcup_d H_d = H \tag{10}$$

The subscript $_d$ runs over the number of domains into which the space has been divided. If the number of domains exceeds the processor count P, a processor will compute more than one domain. If it is less, some processors will be idle or redundant.

By examining Fig. 2 we see that defining data domains $S^D_{d;N}$ is sufficient for the computation $f_{mn} \rightarrow f_{mj}$ to be local, irrespective of the other data distributions. Whether we use a local spectral computational domain S_d or a global one $S_{d;N}$ depends on whether there is

Figure 5: Data domains in the $f_{mj} \rightarrow f_{mn}$ computation.



enough computation in S to outweigh the cost of gathering data across processors to construct $S^D_{d;N}$. If there is not, we would rather perform the computations in S redundantly on each processor. As mentioned earlier, this step will scale perfectly only up to M processors.

Similarly, consulting Fig. 3 – Fig. 5, we may see that the use of domains $F^D_{d;M}$, $G^D_{d;I}$ and $F^D_{d;J}$ for the subsequent steps in the computation loop in Fig. 1 ensures that all global reductions are performed locally. Note especially that the Fourier space is partitioned differently in the forward and backward Legendre transforms. Whether global computational domains $F_{d;M}$, $G_{d;I}$ and $F_{d;J}$ are used depends on the computation to communication ratio at each step. Since the only computations on the Fourier domain are for the transform itself, it is likely that we may use global domains here, especially on loosely-coupled systems. Computation in G is likely to be intensive enough to warrant using local domains G_d. This will

then entail a data gathering step after the gridspace computations are completed, prior to commencing the inverse Fourier transform.

This analysis has been implicitly done with reference to a message-passing environment. It is to be noted, however, that even on a shared-address NUMA machine, this method of partitioning data has much to recommend it. All domains will be local, and the partitioning of data in memory will then naturally be according to the computation on the processor (Culler and Singh 1998). Thus, a global array `f(1:IMAX,1:JMAX)` will instead be dimensioned `f(IS:IE,JS:JE,NDOMAINS)`, where `(IS:IE,JS:JE)` are the local domain bounds, and `NDOMAINS` is the number of domains on the partition.

4 Discussion.

Tests are currently underway to determine the scaling behaviour of this method on both MPP (t3e) and DSM (Origin) platforms. We will report the scalability of the algorithm for various domain configurations and processor counts, and for a range of computational densities in spectral and grid space.

References

Culler, D. E., and J. P. Singh, 1998: *Parallel Computer Architecture: A Hardware/Software Approach*, p. 1100, Morgan Kaufmann Publishers.

Foster, I. T., W. R. Gropp and R. Stevens, 1992: The parallel scalability of the spectral transform method., *Mon. Wea. Rev.*, **120**, 835 – 850.

Held, I. M., and M. J. Suarez, 1994: A proposal for the intercomparison of the dynamical cores of

atmospheric general circulation models., *Bull. Amer. Met. Soc.*, **75**(10), 1825 – 1830.

Orszag, S. A., 1970: Transform method for the calculation of vector-coupled sums: Application to the spectral form of the vorticity equation., *J. Atmos. Sci.*, **27**, 890 – 895.

THE SAME-SOURCE PARALLEL MM5[1]

John Michalakes

Mathematics and Computer Science Division
Argonne National Laboratory
Chicago, Illinois 60439
michalak@mcs.anl.gov
+1 630 252-6646
Fax: +1 630 252-5986

ABSTRACT

With the March 1998 release of the Penn State University/NCAR Mesoscale Model (MM5), the official version of the model (MM5v2 Release 8) now runs on distributed memory (DM) message-passing platforms. Under an IBM-funded effort, source translation and runtime library support minimize the impact of parallelization on the original model source code with the result that the majority of code is line-for-line identical with the original version. Parallel performance and scaling are equivalent to earlier, hand-parallelized versions; the modifications have no effect when the code is compiled and run without the DM option. Supported computers include the IBM SP2, Cray T3E, and Fujitsu VPP. The approach is compatible with shared-memory parallelism, allowing DM/SM hybrid parallelization on distributed memory clusters of SMP. Preliminary results show that scalability on distributed shared memory computers such as the SGI Origin 2000 also benefits from a distributed memory programming model.

1. INTRODUCTION

The Pennsylvania State/National Center for Atmospheric Research Mesoscale Model is a limited-area model of atmospheric systems, now in its fifth generation, MM5 (Grell et al, 1994). It was designed for vector and shared-memory parallel architectures. Two earlier distributed-memory parallel versions of the model code were developed at Argonne National Laboratory -- the Massively Parallel Mesoscale Model (MPMM) and a subsequent Fortran90 implementation, MM90. These were efficient, scalable, more modular and dynamically configurable (Foster and Michalakes, 1993; Michalakes, 1997b) than the source model. Nevertheless, extensive modification for parallelization prevented integration with the official version of MM5. The challenge was to produce a distributed memory parallel version of the model sufficiently close to the original source code that it could be officially adopted, supported, and maintained. This was accomplished March, 1998, with the release of MM5

[1] This work was supported in part by the Mathematical, Informational, and Computational Sciences Division subprogram of the office of Computational and Technology Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

Version 2 Release 8, the first official version of the model to support distributed memory parallelism.

The "same-source" approach to parallelization places an emphasis on preserving the original source code, a critical factor in NCAR's willingness to accept the changes to the official model. The approach employs an application-specific parallel library and a compile-time source translator to automate and hide parallel mechanisms in the code. The Runtime System Library, RSL (Michalakes, 1997c), provides domain decomposition, local address space computation, distributed I/O, and interprocessor communication supporting parallelization of both the solver and mesh refinement code. The Fortran Loop and Index Converter (FLIC (Michalakes, 1997a)), translates at compile-time to generate a parallelized code (that only the compiler sees) from a single version of the source model. The approach is essentially directiveless, requiring only a small amount of information—sufficiently general and concise to fit on the tool's command line—to direct the translation.

The DM-parallel option to MM5 was released as part of the official model in March, 1998, as MM5 Version 2 Release 8. The code is running operationally in real-time forecast mode on an IBM SP at the United States Air Force Weather Agency facility, Offutt Air Force Base, Nebraska. The model is also in use by the U.S. EPA, the California Air Resources Board, and a number of other research, university, and government users in the United States, Europe, and Asia. This paper summarizes issues that arise in parallelization of a weather model and describes the tools-based approach used to parallelizing MM5. Results are evaluated in terms of impact on model source code as well as model performance and scaling.

2. SAME-SOURCE

Architecture-specific coding affects understandability, maintainability, extensibility, reusability, and portability to other, dissimilar architectures. Such coding may manifest itself in how arrays are dimensioned, aligned, and allocated in memory; how loops are nested or otherwise structured (blocked, unrolled, fused); at what level loops are positioned in the subroutine call hierarchy; how iteration is expressed (loops or array syntax); how information is exchanged between subroutines; and, with distributed memory, how communication is implemented. Maintaining separate codes is difficult and time consuming; and because changes and enhancements must be made by hand and tested over all versions, some versions inevitably fall behind. The ability to exploit a range of computer architectures with a single source code provides obvious benefits. If, in addition to a "single-source" the user also wishes to parallelize the code while preserving the pre-existing non-parallel source, the additional constraint of "same-source" is imposed.

Distributed memory programming provides the most general programming model for both portability and scalability, since distributed memory programs adapt trivially to shared memory (while the reverse is not true). Portability through distributed memory programming will best position programs to exploit successive advances in high-performance computer architecture, the latest of which is low-cost high-speed networked configurations of personal computers (Cipra, 1997), a computational option unavailable to shared-memory programs.

Programming for distributed memory provides both portability and scalability. Another emerging architecture is distributed memory configurations of SMP nodes; distributed memory programming is an essential (and non-mutually exclusive) component of an overall strategy to exploit these machines. Finally, on distributed/shared memory architectures – distributed memory machines with additional hardware and software for to support shared-memory programming (e.g. the SGI Origin 2000) -- distributed memory programming may still provides better scaling because locality is explicitely enforced.

Much of the painful low-level detail originally associated with message-passing programming—domain-decomposition, message passing, distributed I/O, and load balancing—has been efficiently encapsulated in application-specific libraries (Hempel and Ritzdorf, 1991; Kohn and Baden, 1996; Michalakes, 1997c; Parashar and Browne, 1995; Rodriguez et al, 1995). These approaches still require modification to the code for iteration over local data, global and local index translation, and distributed I/O. If one is able to design a new model or undertake a major redesign, these issues may be addressed directly in the code, as a number of groups have demonstrated (e.g., ECMWF's IFS and Environnement Canada's MC2 models). However, if a *same*-source and not only single-source implementation is required, additional help is needed.

Source translation removes the remaining difficulties associated with implementing the model efficiently for distributed memory. Further, source translation is applicable to a broader range of performance portability concerns. Loop restructuring, data-in-memory restructuring and realignment, and other manipulations are all effective code transformations for addressing single-processor cache performance, data locality, and communication cost. Source translation and analysis tools also uncover data dependencies in parallel routines (Friedman et al, 1995; Kothari, 1996). Finally, source translators may be used for nonperformance-related code transformations, such as adjoint generation for sensitivities and four-dimensional variational assimilation (Goldman and Cats, 1996). Source translation is a key enabling technology for the single-source development of fully integrated, fully portable models.

3. APPROACH

Parallelizing a weather model for distributed memory parallel computers involves dividing the horizontal dimensions of the domain and assigning the resulting tiles to processors. The code is then restructured to compute only the cells stored locally on each processor (by modifying DO loops and index expressions) and communication is added to exchange data between processors. In an explicit model such as MM5, the communication between processors is essentially nearest neighbor and is used to update extra memory regions around the local partition. Communication is also required to support mesh-refinement in models that support nesting.

Adapting the model to compute over multiple address spaces requires modifying the code to execute only over the local partition on each processor. This involves modification of loops and indices. There are two approaches: either establish that an index expression always

represents a global index (Global View), or establish that the index expression always represents the index of a cell in local memory (Local View). In either case, the actual indexing of the model arrays within the bodies of parallel loops is unaffected; what differs is the expression of the loop ranges themselves, the declaration and storage classes of the decomposed arrays, and the subroutine interfaces. The global view has advantages for new codes while the local view has advantages for a same-source parallelization of a legacy code.

In the global view, ranges of parallel loops in a subroutine are modified to begin and end at the global indices of the first and last cells on the processor. Fortran subrange expressions are used to declare locally sized model arrays whose elements are, nevertheless, globally indexable. The global view allows all index expressions within the subroutine – array indices, tests for boundary conditions, and instances where the value of an index feeds into the computation – to remain as-is. However, since each processor's arrays must be declared using a different subrange (that is, each processor's set of cells starts and ends at different global indices), the mapping of arrays to storage must be dynamic: model arrays must be passed through argument lists or dynamic memory allocation features such as those found in Fortran-90 must be used. Furthermore, local decomposed arrays in the subroutine must also be dynamically allocated using subranges, either explicitly or as stack variables, which is supported in Fortran-90 but not in Fortran-77.

In the local view, as in the global view, loop ranges over decomposed dimensions must be modified, but here they begin and end at local indices of the first and last cell stored on the processor regardless of their position within the global domain. This allows array dimensions to be uniform over processors and avoids the need to overhaul existing data structures. It becomes necessary, however, to translate between local and global meaning of under certain circumstances: loop-invariant index expressions – a constant appears as an index into a decomposed array dimension, for example -- must be converted from global to local. Index expressions that appear in tests for position in the domain – boundary tests, for example – must be converted from local to global. Index expressions whose values feed into the model computation in some way -- computing distances between two points based on their grid indices, for example – most be converted from local to global.

In both the global and local view, modification of looping structures and data declarations are required to adapt the code to distributed memory execution. The global view avoids the need to convert between global and local indexing but requires greater flexibility in declaring and allocating model storage and it requires that data be passed between subroutines through argument lists. The global view should be considered for new codes or codes undergoing major redesign. The local view, on the other hand, requires that indices be treated carefully depending on whether they mean a global or local index, but the local view more easily coexists with static data structures and systematic use of COMMON in existing codes. Because of this, the local view was adopted for the MM5 parallelization.

4. PARALLEL LIBRARY: RSL

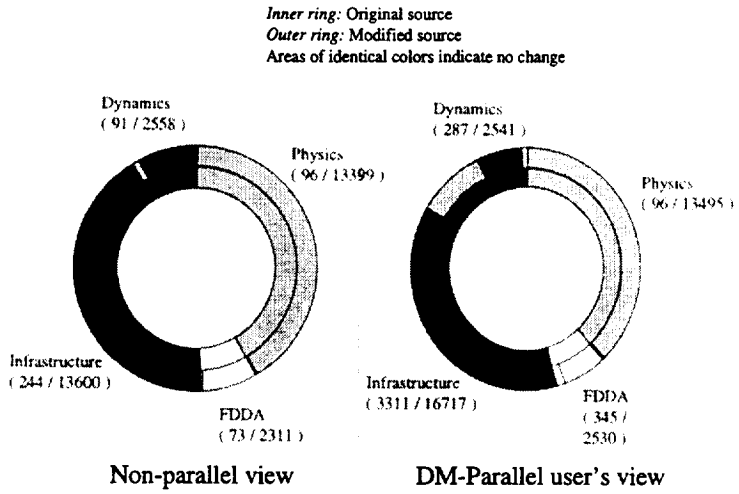


Figure 1 Impact on source

RSL is a parallel runtime system library for implementing regular-grid models with nesting on distributed memory parallel computers. It is used to encapsulate many of the lower-level parallel mechanisms that, otherwise, would require extensive addition and modification to the model source code:

- Domain specification, decomposition over processors, and remapping
- Intra-domain communication (stencil exchanges)
- Inter-domain communication (nest forcing and feedback)
- Local computation on each processor subdomain
- Distributed I/O

RSL and its use in parallelizing MM5 has been described previosly (Michalakes, 1997b,c). Although the library eliminates a large amount of explicit parallel mechanism in the code, its use still requires that the code be modified to compute over local processor subdomains (using either local or global views described above). Therefore, additional encapsulation and automation is required for a fully same-source approach.

5. SOURCE TRANSLATOR: FLIC

Even with parallel libraries, modifications for local address space computation must be made for distributed memory. Hitherto, modifications had been made manually and appeared explicitly as changes to the source code. The same-source approach transfers the responsibility for making these changes to an automatic tool, the source translator, and in the process removes these changes from view of code developers, maintainers, and users. Translations for distributed memory, cache performance, and computational restructuring include:[i]

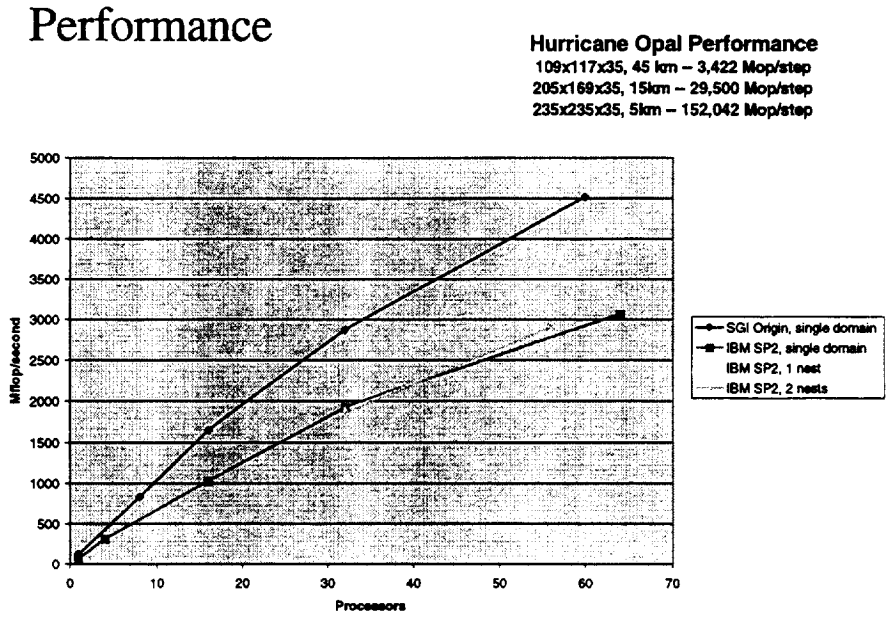


Figure 2. Single and multi-domain performance.

- ✓ Identification and translation of loops over decomposed dimensions,
- ✓ Automatic conversion of loop-invariant expressions indexing decomposed dimensions of model arrays (global to local),
- ✓ Automatic conversion of loop-variable expressions used to test position in the global domain, e.g. for boundary conditions (local to global),
- ❑ Automatic conversion of loop-variable expressions whose values are used within a computation, e.g. computing distance between two points in the logical domain based on their indices (local to global),
- ❑ Automatic insertion of global and local array dimensions into argument lists of subroutines and associated call statements (especially with global view),
- ❑ Automatic disambiguation of expressions used to dimension arrays (local) from loop range expressions,
- ❑ Automatic interprocedural data dependency analysis,
- ✓ Automatic array index reordering,
- ❑ Automatic loop restructuring, and
- ❑ Automatic index algebra to move non-local references to the righthand sides of assignement statements.

We exploit a useful dichotomy in applying source translation to the parallelization task to provide an incremental development path. This is as follows: communication is hard to design but is easy to implement with almost no impact on the source code. Computational restructuring, on the other hand, is straightforward and mechanical, but requires extensive error-prone modification to the source code. Therefore, for parallelizing a code with a

minimum of effort and source code impact, there is an advantage to automating computational restructuring tasks, even if it is necessary to defer automatic dependency analysis for a subsequent phase because some tools are still under development.

The Fortran Loop and Index Converter (FLIC) (Michalakes, 1997a) is a Fortran compiler with a special purpose back-end for generating the modified code. Because it employs full lexical, syntactic, and semantic analysis of the input Fortran, it is able to transform the code with minimal direction. FLIC examines array references within loops and infers which loops are over decomposed dimensions, it uncovers instances where decomposed dimensions are indexed by loop-invariant expressions and generates global to local index translations, and it uncovers instances where expressions of parallel loop variables are used in conditional expressions and generates local to global index translations.

6. RESULTS

The impact on software is extremely small, especially from the point of view of the nonparallel user. Of the 32,000 lines in the model that have been addressed so far, the UNIX diff utility reports 504 lines are different (left half of Figure 1). This view of the code is significant because changes are out of the way of non-parallel users and code developers. One need not even install the DM parallel components, in which case the model is effectively the MM5 code as it exists today.

The right half of Figure 1 shows the parallel user and developer's point of view: the actual number of changes for distributed memory. Physics is virtually unaffected: only 96 of the total 13,495 lines in the parallelized subset are different. Dynamics, which includes communication, is affected slightly more: 287 lines of a total 2,541. Infrastructure, which includes I/O and initialization, effects only 3,300 of a total 16,700. This is due largely to changes relating to distributed I/O, something FLIC does not address. Similarly, the FDDA nudging code is affected because it also includes I/O and several large data reduction operations that FLIC does not, at present, handle.

Figure 2 and Figure 3 show recent preliminary performance results using the MM5 Version 2 Release 8 mode. The results were gathered using the IBM SP at Argonne and the Silicon Graphics Origin 2000 at the University of Illinois (NCSA). Timings on the Origin were performed in dedicated user mode (exclusive access); exclusive access to the processors on the SP was provided by the gang-scheduler. Performance data for other supported platforms – the Cray T3E and Fujitsu VPP300 – were unavailable at the time this paper was prepared. Hurricane Opal data was used to initialize the runs. The box in Figure 2 shows grid sizes, resolutions, and cost in floating point operations (times 10^6) per coarse domain time step for non-nested, singly nested, and doubly nested cases.

Runs using a single domain (non-nested) were conducted to assess performance and scaling of the code in distributed-memory parallel mode on from 1 through 64 processors of the IBM SP (130 Mhz Power2 Superscalar thin nodes with TB3 switch interconnection) and from 1 through 60 processors of the NCSA Origin 2000. The model runs at a rate of 63 Mflop/sec

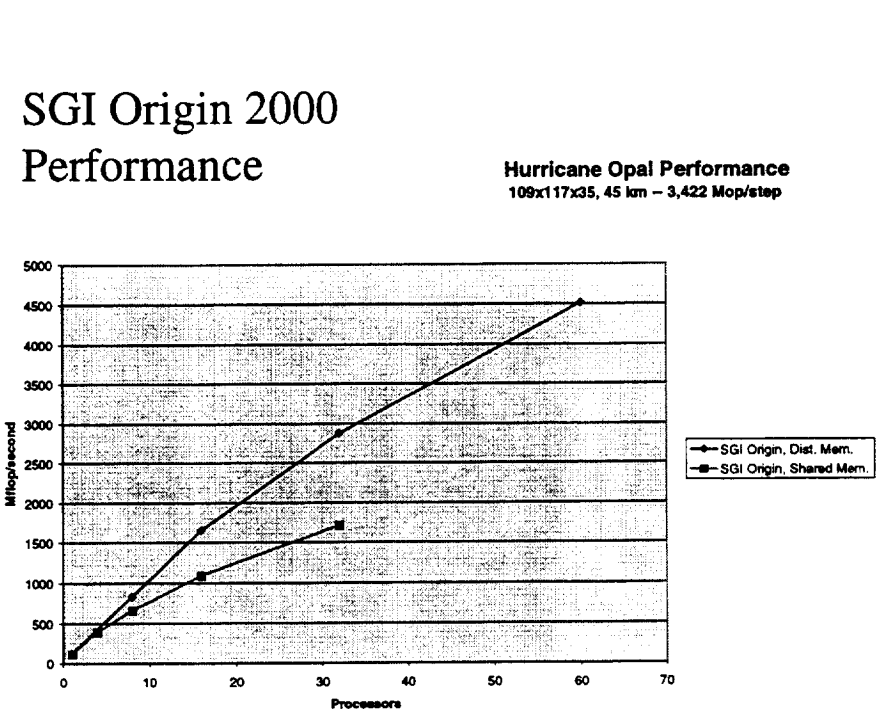


Figure 3. Shared versus distributed memory performance.

on one SP processor versus 118 Mflop/sec on one processor of the origin. On 64 SP processors, the model ran at 3063 Mflop/sec; on the Origin, 4515 Mflop/sec. Parallel efficiency (speedup/P) from 1 to 60 processors on the Origin was 63 percent. IBM SP scaling is super-linear from 1 to 4 processors because of memory effects; therefore a parallel efficiency of 61 percent was calculated from 4 to 64 processors (speedup/P/4). This translates to 14 hours for a 36 hour forecast on one SP processor; 7.7 hours on one processor of the Origin. Running in parallel, the estimated forecast times (exclusive of I/O) are 30 minutes on 64 SP processors and 12 minutes on 60 Origin processors. Singly and doubly nested timings on the SP are shown in Figure 2; performance is actually slightly better because the nested grids are much larger than the coarse domain and therefore run more efficiently.

Figure 3 shows a comparison between shared-memory and distributed memory MM5 runs on the SGI Origin. The distributed memory code is 63 percent efficient from 1 to 60 processors, whereas the shared memory version is only 46 percent efficiency from 1 to 32 processors. Pure distributed memory programs appear able to exploit the low-latency high-bandwidth interconnect to provide scalable performance.

7. CONCLUSION

We have described an effort that will expand the set of architectures that will run the official NCAR version of the MM5, providing the benefit of scalable performance and memory capacity for large problem sizes to users with access to distributed memory parallel computers. The same-source approach uses source-translation technology for adapting MM5,

simplifying maintenance and allowing new physics modules to be incorporated without modification. The fact that MM5 is a fully explicit model is a convenient simplification that may not be available in other models, many of which employ implicit methods in their horizontal dynamics (Baillie et al, 1997). Future work involves adapting and expanding this approach to incorporate other computational techniques, including spectral, semi-implicit, and other methods with non-local data dependencies. Another focus will be on augmenting source code analysis and translation to address cache and other performance portability issues. Same-source tools and techniques provide a reasonable approach to obtaining good performance over the range of high-performance computing options from a single version of the model source code.

REFERENCES

Baillie, C., J. Michalakes, and R. Skålin, 1997: *Regional Weather Modeling on Parallel Computers*, Parallel Computing, (to appear, December 1997).

Cipra, B. A., 1997: "In scientific computing, many hands make light work," SIAM News, 30.

Foster, I. and J. Michalakes, 1993: *MPMM: A Massively Parallel Mesoscale Model*, in Parallel Supercomputing in Atmospheric Science, G. R. Hoffmann and T. Kauranne, eds., World Scientific, River Edge, New Jersey, pp. 354--363.

Friedman, R., J. Levesque, and G. Wagenbreth, 1995: *Fortran Parallelization Handbook*, Applied Parallel Research Inc., Sacramento.

Goldman, V. and G. Cats, 1996: *Automatic adjoint modeling within a program generation framework: A case study for a weather forecasting grid-point model*, in Computational Differentiation, M. Berz, C. Bischof, G. Corliss, and A. Griewank, eds. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1996, pp. 184-194.

Grell, G. A., J. Dudhia, and D. R. Stauffer, 1994: *A Description of the Fifth-Generation Penn State/NCAR Mesoscale Model (MM5)*, Tech. Rep. NCAR/TN-398+STR, National Center for Atmospheric Research, Boulder, Colorado.

Hempel, R., and H. Ritzdorf, 1991: *The GMD Communications Library for Grid-oriented Problems*, Tech. Rep. GMD-0589, German National Research Center for Information Technology.

Kohn, S. R., and S. B. Baden, 1996: *A Parallel Software Infrastructure for Structured Adaptive Mesh Methods*, in Proceedings of Supercomputing '95, IEEE Computer Society Press.

Kothari, S., 1996: *Parallelization Agent for Legacy Codes*, draft technical report, Iowa State University, Ames, Iowa, 1996. See also http://www.cs.iastate.edu/kothari.

Michalakes, J., 1997a: *FLIC: A Translator for Same-source Parallel Implementation of Regular Grid Applications*, Tech. Rep. ANL/MCS-TM-223, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois.

Michalakes, J., 1997b: *MM90: A Scalable Parallel Implementation of the Penn State/NCAR Mesoscale Model (MM5)*, Parallel Computing (to appear); also preprint ANL/MCS- P659-0597.

Michalakes, J., 1997c: *RSL: A Parallel Runtime System Library for Regional Atmospheric Models with Nesting*, Proceedings of the IMA workshop "Structured Adaptive Mesh Refinement Grid Methods," (to appear); also preprint ANL/MCS-P663-0597.

Parashar, M., and J. C. Browne, 1995: *Distributed dynamic data-structures for parallel adaptive mesh-refinement*, Proceedings of the International Conference for High Performance Computing, pp. 22--27.

Rodriguez, B., L. Hart, and T. Henderson, 1995: *A Library for the Portable Parallelization of Operational Weather Forecast Models*, in Coming of Age: Proceedings of the Sixth ECMWF Workshop on the Use of Parallel Processors in Meteorology, World Scientific, River Edge, New Jersey, pp. 148--161.

[i] Checked items are those implemented in FLIC or related translation tools in use within the parallel MM5 effort.

SIMULATIONS OF GRAVITY WAVE INDUCED TURBULENCE USING 512 PE CRAY T3E

Joseph M. Prusa
Iowa State University, Ames, IA
Piotr K. Smolarkiewicz[1]
National Center for Atmospheric Research,[2]Boulder, CO
Andrzej A. Wyszogrodzki
University of Warsaw, Warsaw, Poland

A 3D nonhydrostatic, time-dependent Eulerian/semi-Lagrangian Navier-Stokes solver has been employed to simulate gravity wave induced turbulence at mesopause altitudes. The solver is suitable for modeling a wide range of natural atmospheric flows (broadly documented in the literature). In the present study, the semi-Lagrangian option of the message-passing implementation of the solver was used. This work extends our earlier 2D study reported in the literature to three spatial dimensions while maintaining fine resolution required to capture essential physics of the wave breaking. Present calculations, which would be difficult on standard vector supercomputers, have been performed on the 512 processor Cray T3E machine at the National Energy Research Scientific Computing Center (NERSC) in Berkeley. The physical results of this study are spectacular and clearly demonstrate advantages of highly parallel technologies. In this paper, we briefly outline the physical outcome of the study, while focusing on selected computational aspects of the project. In particular, we compare the relative model performance across several machines (Crays T3E and T3D, Hewlett-Packard Exemplar SPP2000 and Cray PVP systems) using both MPI and Shmem communication software (where applicable).

1. Introduction

In recent years, a number of new machines based on massively parallel processing (MPP) technology have become available for large-scale computations in science and engineering. Among the existing MPP computers, those consisting of hundreds or thousands of processors communicating via explicit message passing implementations of application programs appear particularly competitive with conventional vector supercomputers. On the other hand, there are a number of important yet sufficiently small problems that can be addressed successfully using vector supercomputers, single processor scalar workstations, or even modern PCs.

---

[1] *Corresponding author address*: Piotr K. Smolarkiewicz, NCAR, PO Box 3000, Boulder, CO 80307-3000. E-mail: smolar@ncar.ucar.edu

[2] The National Center for Atmospheric Research is sponsored by the National Science Foundation

In order to best utilize the wide range of computing resources now available for science and engineering, application codes require a high degree of portability between different systems. To appreciate the significance of portability, consider that in the area of computational fluid dynamics, numerical research models usually solve systems of nonlinear partial differential equations on discrete meshes consisting of millions of points over $\mathcal{O}(10^2)-\mathcal{O}(10^4)$ time steps (iterations). The associated computer programs consist of $\mathcal{O}(10^3)-\mathcal{O}(10^5)$ lines of code, and often evolve on a daily basis. Clearly, this makes supporting several versions of the same model cumbersome, expensive, and overall impractical. In this paper we emphasize the portability issue and report on our MPP model performance across several machines representative of the modern computing environment.

The MPP Fortran code adopted for the purpose of this study has been already described in the literature [1, 2]. The underlying solver is an incompressible-type fluid model cast in a curvilinear rotating framework, with a subgrid-scale turbulence parameterization and water substance phase-change processes included. The distinctive aspect of our model [1, 2, 16] is its numerical design which incorporates a two-time-level; either semi-Lagrangian [13] or Eulerian [14], nonoscillatory forward-in-time (NFT) algorithm. The finite-difference approximations to the resulting trajectory-wise or point-wise integrals of the governing fluid equations are at least second-order-accurate. The Eulerian algorithm requires the traditional CFL stability condition, limiting thereby local communications to nearest neighboring points on the mesh; the semi-Lagrangian algorithm admits Courant numbers well exceeding unity and results in irregular communications patterns extending over a number of grid points. In order to take full advantage of MPP systems, the solver has been implemented using a single program multiple data (SPMD) message passing approach. In [1], the authors evaluated the performance of the prototype dynamic core of the model (ideal Boussinesq fluid) for the two optional formulations of the model algorithm and two distinct parallelization approaches (High Performance Fortran, HPF, vs. message passing). In [2], the earlier study was extended to a more complete model (i.e., one including planetary rotation, phase change processes, and subgrid-scale turbulence schemes) suitable for simulating natural atmospheric flows. There, the authors quantified the overall performance of the complete model, as well as the relative performance of its distinct components (transport, elliptic pressure solver, phase-change modeling, subgrid-scale parameterization, etc.), on a distributed memory Cray T3D.

In this paper, we demonstrate a satisfactory performance of the model on a large scientific application, using one of the most complicated options of the model algorithm. As the application addressed is much too large to be executed straightforwardly on other machines available to us, the accompanying studies of the model performance exploit either an abbreviated version of this same experiment, or a less extreme physical scenario of large eddy simulation (LES) of convective planetary boundary layer using the default Eulerian variant of the model algorithm (cf. [18]).

2. Model Description

The numerical model used in this study has been described in [1, 2, 6, 10, 16, 18]. It is representative of a class of nonhydrostatic atmospheric models that solve the anelastic equations of motion in the standard, nonorthogonal terrain-following coordinates. Below we comment briefly on the essential aspects of the model design while referring the reader to the earlier works for further details.

The conservation laws for the dependent variables of the model may be all written in the compact form

$$\frac{\partial \rho^* \psi}{\partial t} + \nabla \cdot (\mathbf{v}^* \psi) = \rho^* F(\mathbf{\Psi}) \ . \tag{1}$$

Here ψ denotes any of the three Cartesian velocity components (u, v, w), the potential temperature, water substance mixing ratios (vapor, cloud water, rain, etc.), as well as turbulent kinetic energy; $\rho^* \equiv \bar{\rho} G$ is the reference (Boussinesq type) density profile premultiplied by the Jacobian of the coordinate transformation (from the Cartesian to the terrain following, time-dependent, curvilinear framework). The advective velocity $\mathbf{v}^* \equiv \rho^* \mathbf{V} \equiv \rho^*(u,\ v,\ \omega)$, with ω denoting the "vertical" component of transformed (contravariant) velocity, satisfies the anelastic mass conservation law

$$\frac{\partial \rho^*}{\partial t} + \nabla \cdot \mathbf{v}^* = 0 \ . \tag{2}$$

Note that the time derivative must be retained in (2) because of the time variation in the coordinate transformation [10]. The associated $F(\mathbf{\Psi})$ terms on the rhs of (1) are, in general, functionals of the vector $\mathbf{\Psi}$ of all dependent variables ψ, and they represent the sum of the resolved and subgrid-scale parts of the total forcings. In the momentum equations, the resolved terms include pressure gradient forces, Coriolis accelerations, buoyancy force, as well as wave absorbing devices in the vicinity of open boundaries. In the thermodynamic equations, the resolved terms include heat and moisture sink/sources due to the phase changes of water, and the wave absorbers near the boundaries. The subgrid-scale (SGS) forcing terms are fairly complex but standard. We employ a turbulence model based on the prognostic turbulent kinetic energy equation [11] or, optionally, its abbreviated version—the celebrated Smagorinsky model.

The integration of the discrete equations over a time-step uses a regular unstaggered mesh. We write the finite-difference approximations to (1) in the compact form

$$\psi_{\mathbf{i}}^{n+1} = LE(\widetilde{\psi}) + 0.5 \Delta t F_{\mathbf{i}}^{n+1} \ . \tag{3}$$

Here, LE denotes either the advective semi-Lagrangian or flux-form Eulerian NFT transport operator; $\widetilde{\psi} \equiv \psi^n + 0.5 \Delta t F^n$; and indices $\mathbf{i}$ and n have the usual meaning of the spatial and temporal location on a (logically) rectangular Cartesian mesh.

Completion of the model time step requires the F^{n+1} values of forcings in (3). Gravity wave absorbers, Coriolis accelerations, condensation, and pressure gradient forces are treated implicitly; whereas subgrid-scale terms, and slow phase-change tendencies (such as rain formation or evaporation [6]) are treated explicitly (i.e., F^{n+1} is predicted from earlier values of dependent variables). The implicitness of the pressure gradient forces is essential as it enables projecting the preliminary values $LE(\widetilde{\psi})$ onto solutions of the continuity equation (2), [4]. Here, it requires a straightforward algebraic inversion of the linear system composed of equations (3), and the formulation of the boundary value problem for pressure implied by the continuity constraint (2). The resulting elliptic equation is solved (subject to appropriate boundary conditions) using the generalized conjugate-residual approach—a preconditioned nonsymmetric Krylov solver (see [15-17] for further details). The numerical stability of computations is controlled by proper limiting of Courant and Lipschitz numbers $\mathrm{C} = \| \Delta t \mathbf{V}/\Delta \mathbf{X} \|$ and $\mathrm{L} = \| \Delta t(\partial \mathbf{V}/\partial \mathbf{x}) \|$, respectively, for the Eulerian and semi-Lagrangian variants of the model.

3. Parallelization versus portability strategy

In [1], we have evaluated the relative merits of message passing and HPF strategies of parallelization. Overall, we have concluded that the message passing code runs 2.5 and 1.8 times faster (on Cray T3D) than the HPF code, respectively, for the Eulerian and semi-Lagrangian versions of the model. Consequently, we settled on a message passing approach. We used a two dimensional horizontal decomposition of the grid; and explicitly dimensioned each array to contain a subgrid of the total array plus extra space for a copy of the neighboring processors' boundary cells. This is a common technique (cf. [7]) where the extra boundary points are often referred to as "halo cells" or "halos". They are used to minimize communications needed when finite difference operations are performed. The number of halo cells depends on the local stencils used in the model algorithm and on the maximum Courant number. In simulations reported here, $\mathrm{C} \leq 3$. When necessary, the halo cell information is updated by having each processor exchange information with its neighbors. This communication process is further economized by admitting only a partial update of halos with their selected portions being exchanged between the processors as implied by the finite-difference algorithms employed. Reduction operations such as sums and extrema, unavoidable in fluid models, require exchanging informations globally between all processors.

To exchange messages between processors, in general, we use the most portable and widely supported MPI (message-passing-interface) standard. However, on Cray's T3D and T3E machines we optionally employ Shmem (shared-distributed memory data-passing support) library routines.

In order to facilitate portability of the code, we use these same halo-update subprograms on the distributed or shared-memory parallel architectures, as well as on a single processor machines. On single-processor and shared-memory platforms, all updates

are elementary. They employ one processor for the total domain dimension with halo used to set appropriate conditions at the domain boundaries. In this case, there is no need for an explicit message-passing protocol, and only selected parts of total arrays are rewritten to halo cells on this same processor.

In regards to the portability issue, input/output (I/O) operations raise some serious concerns. In general, outputed fields should be independent of the machine size and number of parallel processors used in simulations. Files written by programs running on N processors should be readable to applications running on M processors. This is convenient for debugging and is especially important for postprocessing (e.g., diagnostics, visualizations, etc.) of large computing projects. Furthermore, the output files must be also readable on different platforms with different binary file formats (Cray floating-point, Cray 64-bit IEEE, standard 32-bit IEEE, etc.). In our code, one processor performs all I/O communication operations by collecting and distributing arrays between other processors. Relative efficiencies of these I/O operations depend on the particular computer at hand and are important to the overall model performance. Keeping the total grid array on one processor does have the disadvantage of limiting the size of the application. But this is more than compensated by simplicity in coding, and seems to offer an optimal performance.

4. Physical problem

Our test problem for the Cray T3E was to simulate the evolution of an internal gravity wave packet generated by a narrow, 2D squall line at tropopause levels and it's subsequent breaking near the mesopause. The squall line disturbance was Gaussian in its spatial-temporal evolution, with a maximum forcing amplitude of 200 m at $t = 2$ h. The basic state was one of uniform zonal wind ($u_o = -32$ ms^{-1}), stability (with Brunt-Väisällä frequency $N = 0.02$ s^{-1}), and density scale height ($H = 6.63$ km). Through a dispersive mechanism this basic state and forcing favor the development of a monochromatic, 2D primary (convective) instability with near unit aspect ratio. For a comprehensive description of the basic state, run set up, and results and analysis of the ensuing convective instability, see [10]. The problem is of interest for at least two reasons. First, the middle atmosphere is known to be far from radiative equilibrium at mesopause altitudes and wave forcing is the main factor behind this phenomenon [5]. Determination of the extent to which gravity wave breaking in particular is responsible has great relevance to a complete understanding of the process and it's parameterizations. Second, numerical simulation of turbulence is of considerable theoretical interest. The wavebreaking in this study generated a highly inhomogeneous, anisotropic turbulence. This turbulence developed completely from a very smooth linear wavefield in accord with the physics of a wave packet propagating into a very deep model atmosphere—it was not initialized according to any *a priori* turbulence model nor constrained by domain size (which can limit wave-wave interactions [12]). Some idea of the inhomogeneity of the wavefield can be gleaned from Fig. 1a which

shows a contour density plot of the potential temperature (θ) field. The vertical plane of this plot is perpendicular to the zonal flow. It shows the wavefield to be homogeneous in the spanwise direction (left to right) but inhomogeneous in the vertical (note that the complete altitude range is $15 \leq z \leq 125$ km; the regions above and below that shown in Fig. 1a are very smooth and characterized by constant stratification). Similar inhomogeneity occurs in the zonal direction [10].

The computational grid consisted of $544 \times 80 \times 291$ points with a resolution of 380 m. To save computer resources, the problem was executed in 2D on a $544 \times 1 \times 291$ grid until 120 minutes of the physical time. At 120 minutes, the 3D domain was created by repeating the solution in the spanwise direction y, and seeding the buoyancy field with a small amplitude (1 % of the basic state) white noise. Further computations continued in five minute portions of physical time. The lateral zonal and spanwise boundaries were periodic with lateral zonal sponges. A specially tuned vertical sponge was also employed, such that it approximated the effects of atmospheric viscosity. No explicit sub-grid scale viscosity was employed in this simulation. Instead, the removal of energy at the grid scale was effected with the monotonicity option in the interpolator. This option invokes a topological constraint whereby no two streamtubes are allowed to intersect. Essentially energy is removed at the grid scale to the extent needed to avoid local negative entropy production. This corresponds well with the Kolmogorov microscale which is the same order of magnitude as the grid size at the initial altitude of breaking. The time chosen for 3D seeding was carefully selected based upon data generated with earlier 2D [10] and 3D [9] experiments. The run was terminated at 180 minutes because at this point breaking had consumed the zonal extent of the domain.

The evolution of the turbulence was assessed by examining 1D energy spectra computed from θ (approximate equipartition occurs even during wave breaking [9]). In order to place the averaging regions in the turbulence, the zonal and spanwise spectra were computed from the average of local field data centered at 100 km altitude, while vertical spectra fields were centered at a zonal location of -35 km. Furthermore, the spanwise averages were restricted to the zonal range $-60 \leq x \leq -30$ km. All raw spectra were Hamming-Tukey smoothed, which acts to minimize the aliasing effects of the finite domain size, [3]. The inhomogeneity of the wavefield caused severe aliasing problems in the zonal (and vertical) spectra at intermediate to high wavenumbers. In particular, the localization of high wavenumber features to a ~ 50 km zonal (~ 30 km vertical) sub-domain caused (i) severe Gibb's oscillations, and (ii) excess power at the highest wavenumbers. These spectra were further processed by (i) integrating them over a wavenumber variable bandwidth from $k = 1$ km^{-1} to the Nyquist wavenumber of 8.27 km^{-1}, and(ii) employing a differential correction algorithm. This algorithm corrected the full-domain spectra at the highest wavenumbers so that they would show the same power law tendencies as an unaliased, sub-domain spectra computed from fields that lay completely within the turbulence. The zonal spectra shown in Fig. 1b depict both the highest and lowest wavenumber features with fidelity.

Figure 1b shows the evolution of the zonal energy spectra. At 125 minutes energy is concentrated at $k = 0.40$ km^{-1} ($\ln(k) = -0.9$), corresponding to $\lambda_x = 15.5$ km. This fundamental is due to linear growth of the gravity wave packet as it ascends. At $k = 0.80$ km^{-1} ($\ln(k) = -0.23$) a much weaker second harmonic of the fundamental can also be seen. This second harmonic is due to nonlinear effects, and is a harbinger of the primary convective instability which is about to occur. For the given basic state, linear waves have an evanescent limit at $k = 0.62$ km^{-1}, this corresponds to the very sharp drop off between the fundamental and second harmonic. At later times linear dispersion causes the fundamental to broaden and peak at lower wavenumbers (longer waves are slower and take more time to propagate upwards [10]). With the onset of vigorous wave overturning, a buoyancy subrange [19] with a slope of -3 appears just upscale of the fundamental (obvious by 140 minutes). At the highest wavenumbers there is negligible energy until the primary instability occurs. With the onset of a secondary (3D) instability, a tendency towards a -5/3 slope can be seen. The critical buoyancy wavenumber that separates the two regimes decreases from $k_b = 4.0$ to 1.8 km^{-1} as t increases from 140 to 180 minutes, respectively. This compares favorably with the earlier J90 results which yielded $k_b = 2.1$ km^{-1} at 150 minutes [9]. The experimental value of k_b may also be compared with the scaling result, $k_b \sim N^3/\epsilon_o \sim 1.6$ km^{-1} [19], where ϵ_o is the turbulence dissipation rate. Finally Fig. 1b clearly shows another -5/3 power law regime at the lowest wavenumbers at earlier times (125 to 145 minutes). This is consistent with a 2D reversed energy cascade that is transferring energy into the zonal mean fields [8]. The Eliassen Palm flux divergence has its maximum value precisely in this time interval, of order 0.02 ms^{-1}, at breaking altitudes. After 150 minutes, the energy spectra flatten out at the lowest wavenumbers. At this point wave breaking has disrupted the linear wave field sufficiently that it lacks the large scale coherence needed to effectively modify the zonal average state. Vertical spectra (not shown) show very similar evolutionary tendencies, with the only significant difference being a lack of the -5/3 power law regime at the lowest wavenumbers. Spanwise energy spectra (not shown) show very different evolutionary tendencies, however. The spectrum at 125 minutes is quite flat and 15 orders of magnitude below the fundamental of the zonal spectra. Growth of spanwise energy is negligible for the first 10-15 minutes after the 3D seeding. In the next 5-10 minutes spanwise spectral energy explodes as the secondary instability undergoes a period of exponential growth. An inertial subrange, characterized by a -5/3 power law appears at the highest wavenumbers. As t continues to increase, this subrange expands to lower wavenumbers, until at 180 minutes, most of the spectrum lies within it.

5. Model performance results

The experiment described in the preceding section has been performed on the 512 processors Cray T3E machine at NERSC. Table 1 outlines the history of its computational cost versus the overall model performance (measured by the wallclock time) as functions of the simulated physical time; time step Δt; number of time steps

Nt and average number of iterations $N_{\rm it}$ in the elliptic Krylov solver (per timestep) per 5 minute portion of the experiment. In addition to summarizing elementary aspects of the model efficiency, this table illustrates an important point that the overall model performance (as well as the relative performance of various model components such as advective transport, pressure solver, etc.; see [2] for a discussion) is an elusive entity, as it is a complicated function of the simulated flow. Consider, for instance, that at the onset of vigorous wave breaking at 145 min., the accuracy arguments [10, 13] dictate halving the time step. Yet, as the flow becomes more quiescent following the onset of breaking, the elliptic solver converges (see [17] for a discussion of the convergence criteria in function of the model algorithm) using only a third as many iterations.

TABLE 1. Gravity-wave breaking experiment on 512 PE Cray T3E. The history of the wallclock and CPU time (sec.) as functions of the simulated physical time (min.), time step Δt (sec.), and average number of iterations in the elliptic pressure-solver per 5-min. portion of the experiment (only every second portion is shown).

physical time	Δt	Nt	$N_{\rm it}$	wallclock time	User CPU
125-130	5	60	32	1156	583159
135-140	5	60	31	1123	566583
145-150	2.5	120	19	1500	757776
155-160	2.5	120	15	1379	696690
165-170	2.5	120	12	1278	645268
175-180	2.5	120	11	1212	611210

The gravity-wave-breaking experiment was much too large to be used as a benchmark for any systematic performance studies across various machines. Also, our limited resources at NERSC precluded any 'lavishness' and left little room for additional tests beyond the production runs. As a result, the relative performance issues were addressed using either a 2D variant of the experiment, or our earlier LES calculations of convective boundary layers, using the Eulerian variant of the model with the nonoscillatory option of the MPDATA algorithm for the LE operator in (3), [2, 18].

The results of the model-performance analysis are gathered in Tables 2-4. They further demonstrate that overall model performance is not only a function of the flow but also depends upon the machine, communication software, compiler options, model algorithms, and size of the problem. Table 2 describes the machines used in this study, whereas Table 3 addresses the scalability issue exploiting a 2D variant of the gravity-wave experiment ($544 \times 1 \times 291$ mesh with $\Delta t = 5$ s and $Nt = 1800$). These simulations were performed on 16 and 32 processors only. On HP and T3D, the resulting speedups are quite good regardless of the communication software employed (consistent with our earlier experience [1, 2]), but they appear relatively poor on T3E-900 using either

communication software. A similar analysis but using a larger 3D problem of the LES boundary-layer experiment (Table 4) shows, for the larger number of Cray processors, more respectable results closer to ~ 1.8 value concluded in our earlier studies [1, 2].

TABLE 2. Machines employed in model performance studies. Columns two to five show the number of processors, location of the machine, nominal memory, and typical measured performance (Mflops/PE), respectively. *Experiments performed in the double 64-bit precision, to match the Cray standard. †Interdisciplinary Center for Mathematical and Computational Modeling, University of Warsaw, Poland.

machine	# PE	location	memory	performance
T3E-900	512	NERSC	256 MB/PE	150-300
T3E-600	32	ICM†	128 MB/PE	100-200
T3D	128	NCAR	64 MB/PE	15
HP*	64	NCAR	8 GB	120
Cray J932se	24	NCAR	8 GB	60
Cray J916	16	NCAR	2 GB	60

TABLE 3. Scalability results using 2D semi-Lagrangian simulation of the gravity wave. The "Oi" symbol in the first column refers to the compiler optimization level. The second column specifies the communication software employed. Columns 3 and 5 show the wallclock time (sec.) of the entire experiment, whereas column 4 shows the resulting speedup ($\nearrow$).

machine	comm. soft.	16 PE	$\nearrow$	32 PE
HP O1	MPI	22354	1.93	11571
HP O2	MPI	7192	1.91	3758
T3D	Shmem	9911	1.88	5266
T3E-900	Shmem	4365	1.58	2768
T3E-900	MPI	5325	1.50	3542

The observed discrepancies are not necessarily surprising. Since the simulated turbulent flows are highly chaotic and unpredictable, the model algorithm can react to even such minor changes in the code setup as the number of processors or the compiler employed.[3] This sensitivity is insignificant insofar as the physical issues are concerned, but it can quite substantially affect model performance.

[3] The different execution of sums inherent in elliptic Krylov solvers can affect both the evaluated pressure field and the number of the iterations required.

TABLE 4. Scalability results using 3D Eulerian simulations of convective boundary layer. #The equivalent semi-Lagrangian run included for comparison. *J90 autotasking, shared (as oppose to dedicated) mode, runs; †24 PE run.

machine	comm. soft.	8 PE	↗	16 PE	↗	32 PE	↗	64 PE
HP O1	MPI	24308	1.89	12836	1.50	8542	1.60	5325
HP O2	MPI	8912	1.65	5395	1.63	3307	1.44	2299
HP O3	MPI			6516	1.64	3969	1.43	2777
HP O4	MPI			6463	1.54	4179	1.46	2859
T3D	Shmem	16622	1.87	8866	1.61	5492	1.71	3164
T3D#	Shmem			15569	1.78	8751	1.79	4876
T3E-900	Shmem			3199	1.43	2236	1.77	1260
T3E-900	MPI			3555	1.43	2476	1.63	1516
T3E-600	Shmem	9127	1.92	4735				
T3E-600	MPI	7471	1.87	3995				
J932se*	none	14087		8545		4987†		
J916*	none	7696		7039				

Table 4 contains a number of hints useful for the interested practitioner. We draw attention to a few points that are especially noteworthy. The results from autotasking runs on Crays J932se and J916 depend on the actual state of the machine, so they should be viewed only as examples of possible overall performances; however, in our experience, the wallclock times attained with maximal number of processors are representative. The semi-Lagrangian run is about 50% more expensive than the Eulerian run at the same Δt (here $C \lesssim 1$). However, in our breaking gravity wave problem, much larger time steps are used (with $C \lesssim 3$) and, *more important*, the semi-Lagrangian algorithm is more accurate as it treats equally the incompressible and compressible numerical regimes of flow, dictated by the specified time-dependency of the problem geometry.

6. Remarks

The horizontal grid decomposition employed for the message-passing MPP implementation of our model was dictated, in essence, by the physics of natural geophysical flows that makes the vertical (gravity) direction distinct. Coincidently, it has a purely computational advantage of admitting efficient applications of this same

model algorithm and code design on different types of machines including distributed- and memory-shared as well as single-processor architectures.

Although our MPP model has been designed to run efficiently on the distributed memory architectures, it appears to perform reasonably well on standard vector supercomputers. Consider that the original version of the same model, optimized for the shared-memory Cray vector machines, achieves on average $65-90$ Mflops per processor on J90s, depending on the number of processors and the application addressed; whereas, the MPP code is only slightly slower on these machines with its speed falling in the range of $60-85$ Mflops/PE.

Regardless of all the objective model-performance measures discussed in this paper, the single most important outcome of this study cannot be overstated: Our earlier, one order of magnitude smaller but otherwise analogous to the 512 PE Cray T3E gravity-wave simulation, experiments performed in the autotasking mode on 24 processor Cray J90 at NCAR, used to take several days (this includes waiting in economy queues) to accomplish simulation of 5 minutes of the physical time. Present experiments, on the order of magnitude larger grid, were executed essentially overnight for the same 5 minutes period of simulated physical time!

Acknowledgements. We thank William Anderson for his advice on parallelization issues and comments on an earlier version of the manuscript. This work has been supported in part by the Department of Energy "Computer Hardware, Advanced Mathematics, Model Physics" (CHAMMP) research program. The use of the 512 PE Cray T3E at NERSC and of the 32 PE Cray T3E at ICM is gratefully acknowledged.

REFERENCES

1. Anderson, W.D., and Smolarkiewicz, P.K., A comparison of High Performance Fortran and message passing parallelization of a geophysical fluid model. In *Parallel Computational Fluid Dynamics: Algorithms and Results Using Advanced Computers*, Schiano et al. Editors, Elsevier Science, 384 (1997).

2. ____, V. Grubišić, and P.K. Smolarkiewicz, Performance of a massively parallel 3D non-hydrostatic atmospheric fluid model, in *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA'97*, Las Vegas, Nevada, USA, June 30-July 3, Ed. H.R. Arabnia, CSREA, 645 (1997).

3. Bath, M., *Spectral Analysis in Geophysics.* Developments in Solid Earth Geophysics, **7**, Elsevier Scientific Publishing Co., Amsterdam, 563p (1974).

4. Chorin, A.J., Numerical solution of the Navier-Stokes equations. *Math. Comp.*, **22**, 742 (1968).

5. Garcia, R.R, and S. Solomon, The Effects of Breaking Gravity Waves on the Dynamics and Chemical Composition of the Mesosphere and Lower Thermosphere. *J. Geophys. Res.*, **90**, 3850 (1985).

6. Grabowski, W.W., and Smolarkiewicz, P.K., On two-time level semi-Lagrangian modeling of precipitating clouds. *Mon. Wea. Rev.*, **124**, 487 (1996).

7. Johnson, K.W., Bauer J., Riccardi G.A., Droegemeier K.K., and Xue M., Distributed processing of a regional prediction model, *Mon. Wea. Rev.*, **122**, 2558 (1994).

8. Kraichnan, R.H., Inertial Ranges in Two-Dimensional Turbulence. *Phys. Fluids*, **10**, 1417 (1967).

9. Prusa, J.M., Garcia, R.R., and P.K. Smolarkiewicz, Three-Dimensional Evolution of Gravity Wave Breaking in the Mesosphere. *11th Conf. Atmos. Ocean. Fluid Dynamics*, Tacoma, WA, J3 (1997).

10. Prusa, J.M., Smolarkiewicz, P.K., and Garcia, R.R., On the propagation and breaking at high altitudes of gravity waves excited by tropospheric forcing. *J. Atmos. Sci.*, **53**, 2186 (1996).

11. Schumann, U., Subgrid Length-Scales for Large-Eddy Simulation of Stratified Turbulence. *Theoret. Comput. Fluid Dynamics*, **2**, 279 (1991).

12. Scinocca, J.F., The Mixing of Mass and Momentum by Kelvin-Helmholtz Billows. *J. Atmos. Sci.*, **52**, 2509 (1995).

13. Smolarkiewicz P.K., and Pudykiewicz, J.A., A class of semi-Lagrangian approximations for fluids. *J. Atmos. Sci.*, **49**, 2082 (1992).

14. ____, and Margolin, L.G., On forward-in-time differencing for fluids: Extension to curvilinear coordinates. *Mon. Wea. Rev.*, **121**, 1847 (1993).

15. ____, and ____., Variational solver for elliptic problems in atmospheric flows. *Appl. Math. & Comp. Sci.*, **4**, 527 (1994).

16. ____, and ____., On forward-in-time differencing for fluids: An Eulerian/semi-Lagrangian nonhydrostatic model for stratified flows. *Atmos.-Ocean Special*, **35**, 127 (1997).

17. ____, Grubišić, V., and Margolin, L.G., On forward-in-time differencing for fluids: Stopping criteria for iterative solutions of anelastic pressure equations. *Mon. Wea. Rev.*, **125**, 647 (1997).

18. ____, and Margolin, L.G., MPDATA: A Finite-Difference Solver for Geophysical Flows, *J. Comput. Phys.*, **140**, 459 (1998).

19. Weinstock, J., Theoretical Gravity Wave Spectrum in the Atmosphere: Strong and Weak Wave Interactions. *Radio Sci.*, **20**, 1295 (1985).

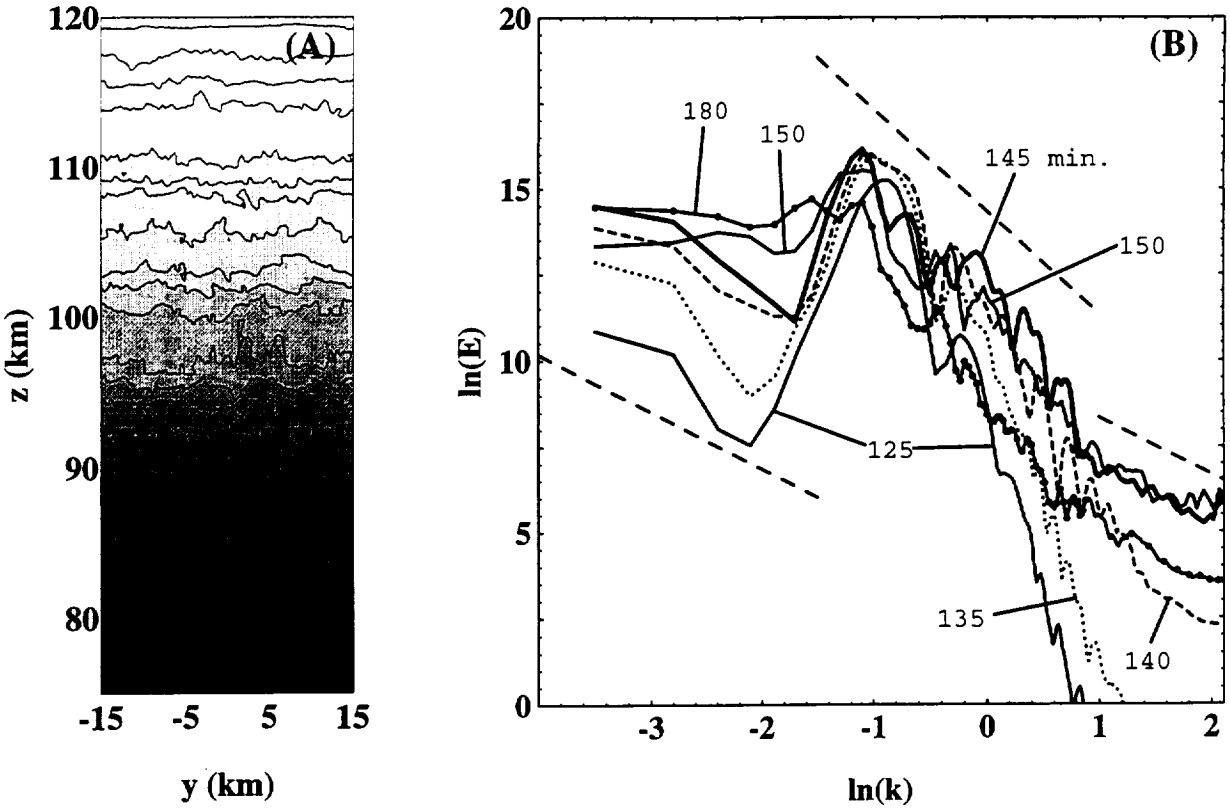


Figure 1. Results from the wavebreaking experiment: (a) contour density plot of $\ln(\theta)$ in vertical yz (spanwise) plane at zonal location x = -35 km at t=155 minutes showing region of vigorously breaking waves; (b) time evolution of zonal spectral energy from 125 to 180 minutes (straight dashed lines have slopes of -5/3 and -3).

An Overview of the Physical-space Statistical Analysis System Development at the Data Assimilation Office

J. Guo
SAIC/General Sciences Corporation, Laurel, MD
mailstop: Data Assimilation Office, NASA/Goddard Space Flight Center
Code 910.3, Greenbelt, MD 20771
mailto: guo@dao.gsfc.nasa.gov
office: +1 301 805-8333
fax: +1 301 805-7960

J. W. Larson and **P. M. Lyster**
Department of Meteorology and Earth System Science Interdisciplinary Center
University of Maryland, College Park, MD

Atmospheric data assimilation is a contemporary scientific discipline for studying the state of the atmosphere. Its object is to produce an optimal and physically consistent four dimensional estimate of the state of the atmosphere using observations from highly diverse sources available irregularly in space and time. Typical methodologies of data assimilation use numerical prediction models, such as General Circulation Models (GCM), and objective analysis systems, such as the Physical-space Statistical Analysis System (PSAS).

PSAS is one of the central components of the Data Assimilation System being developed at the Data Assimilation Office (DAO), NASA/Goddard Space Flight Center, for NASA's Earth Observing System (EOS). It has been designed to replace the traditional analysis scheme used in the earlier version of the DAO's Data Assimilation System with a state-of-art global statistical analysis scheme capable of handling operationally a huge and diverse observational data flow, and to provide the basic computational infrastructure for the DAO research activities using advanced error covariance models.

To realize its scientific goals, PSAS is being designed to overcome several computational obstacles, including the computational complexity of solving the statistical analysis equations for error covariance matrices typically of order 100,000 $\times$ 100,000, and the software complexity of supporting advanced error covariance models accessing a heterogeneous array of information resources. At the same time, we are also actively exploring the path of implementing the technology of distributed parallel computing to PSAS in an operational environment.

Implementing a message passing parallel computing paradigm into an existing yet developing computational system as complex as PSAS is in many ways a nontrivial problem. The technical challenges include issues such as reassessing the system's computational and operational requirements with appropriate solutions, planning and managing a gradual system development path with parallel software development efforts, and particularly, designing software structures supporting planned but often unspecified future scientific extensions.

Incorporating Parallel Computing into the Goddard Earth Observing System Data Assimilation System (GEOS DAS)

Jay Larson
NASA Data Assimilation Office (DAO)
7501 Forbes Blvd., Suite 200, Seabrook, MD 20706
email: jlarson@dao.gsfc.nasa.gov
phone: +1 301 805-8334 fax: +1 301 805-7960

Atmospheric data assimilation is a method of combining actual observations with model forecasts to produce a more accurate description of the earth system than the observations or forecast alone can provide. The output of data assimilation, sometimes called the *analysis*, are regular, gridded datasets of observed and unobserved variables. Analysis plays a key role in numerical weather prediction and is becoming increasingly important for climate research. These applications, and the need for timely validation of scientific enhancements to the data assimilation system pose computational demands that are best met by distributed parallel software.

The mission of the NASA Data Assimilation Office (DAO) is to provide datasets for climate research and to support NASA satellite and aircraft missions. The system used to create these datasets is the Goddard Earth Observing System Data Assimilation System (GEOS DAS). The core components of the the GEOS DAS are: the GEOS General Circulation Model (GCM), the Physical-space Statistical Analysis System (PSAS), the Observer, the on-line Quality Control (QC) system, the Coupler (which feeds analysis increments back to the GCM), and an I/O package for processing the large amounts of data the system produces (which will be described in another presentation in this session).

The discussion will center on the following issues: the computational complexity for the whole GEOS DAS, assessment of the performance of the individual elements of GEOS DAS, and parallelization strategy for some of the components of the system.

I/O Parallelization for the Goddard Earth Observing System Data Assimilation System (GEOS DAS)

R. Lucchesi, W. Sawyer†, L. L. Takacs, P. Lyster†, J. Zero
Data Assimilation office
NASA/GSFC, Code 910.3
Greenbelt MD, 20771
Tel: +1 301 286-9084
Fax: +1 301 286-1754
Email: lucchesi@dao.gsfc.nasa.gov
† Additional affiliation: University of Maryland, College Park, 20742

May 20, 1998

Introduction

The National Aeronautics and Space Administration (NASA) Data Assimilation Office (DAO) at the Goddard Space Flight Center (GSFC) has developed the GEOS DAS, a data assimilation system that provides production support for NASA missions and will support NASA's Earth Observing System (EOS) in the coming years. The GEOS DAS will be used to provide background fields of meteorological quantities to EOS satellite instrument teams for use in their data algorithms as well as providing assimilated data sets for climate studies on decadal time scales. The DAO has been involved in prototyping parallel implementations of the GEOS DAS for a number of years and is now embarking on an effort to convert the production version from shared-memory parallelism to distributed-memory parallelism using the portable Message-Passing Interface (MPI).

The GEOS DAS consists of two main components, an atmospheric General Circulation Model (GCM) and a Physical-space Statistical Analysis System (PSAS). The GCM operates on data that are stored on a regular grid while PSAS works with observational data that are scattered irregularly throughout the atmosphere. As a result, the two components have different data decompositions. The GCM is decomposed horizontally as a checkerboard with all vertical levels of each box existing on the same processing element(PE). The dynamical core of the GCM can also operate on a rotated grid, which requires communication-intensive grid transformations during GCM integration. PSAS groups observations on PEs in a more irregular and dynamic fashion.

I/O Requirements

A primary requirement of the shared-memory GEOS DAS is to provide real-time support for EOS instrument teams after the EOS AM-1 satellite launch scheduled for late 1998. The first delivery of the MPI-based GEOS DAS will have to meet requirements imposed by this mission. One requirement of the I/O system is the generation of a suite of output streams [1] without impacting the performance of the system. The data files must be in HDF-EOS format to allow subsequent distribution to instrument teams and users. There is also an internal requirement for the GEOS DAS to assimilate 30 days of data in one calendar day as part of generating long-term reanalysis data sets. This is the requirement that places the greatest performance demands on the I/O system. Given that one day of assimilation must occur in 48 minutes, very little time can be spent waiting for I/O processes. The expected input and output data volumes for AM-1 support and reanalysis are specified in the following table.

Input	Frequency	Size/day	Format	Module
Boundary Conditions	24 hr	5 MB	GrADS	GCM
Restart	initial	250 MB	64 bit	GCM
Observations	6 hr	50 MB	ODS	PSAS
User Input	initial	1 MB	text	Both
Output				
History	3 or 6 hr	1200 MB	HDF-EOS	GCM
Obs with QC	6 hr	50 MB	ODS	PSAS
Restart	24 hr	250 MB	64 bit	GCM
Diagnostic Info	continuous	1 MB	text	Both

Table 1: This table shows the input and output streams for the GEOS DAS in support of the EOS AM-1 launch. All data sizes are for one day of assimilation (not a calendar day).

It is clear from Table 1 that I/O requirements of the system are dominated by the GCM as roughly 70% of I/O is output from the GCM's "History" component. The History component reads user-generated namelist files that defines some number of output streams and a collection of model state and diagnostic variables for each stream. As the GEOS DAS assimilation moves forward in time, these history streams are written to disk at frequencies defined by the user. History has the potential to be a significant bottleneck in both the shared-memory and distributed-memory systems. This is especially true because the History module does more than just write data to a file. It must convert the data from the internal computational space to a grid space and data format useful for the user. This often requires grid transformations and other formatting which may be computationally expensive. Our highest priority for the GEOS Parallel I/O Subsystem (GPIOS) is to design a portable and efficient MPI implementation of the GCM History module.

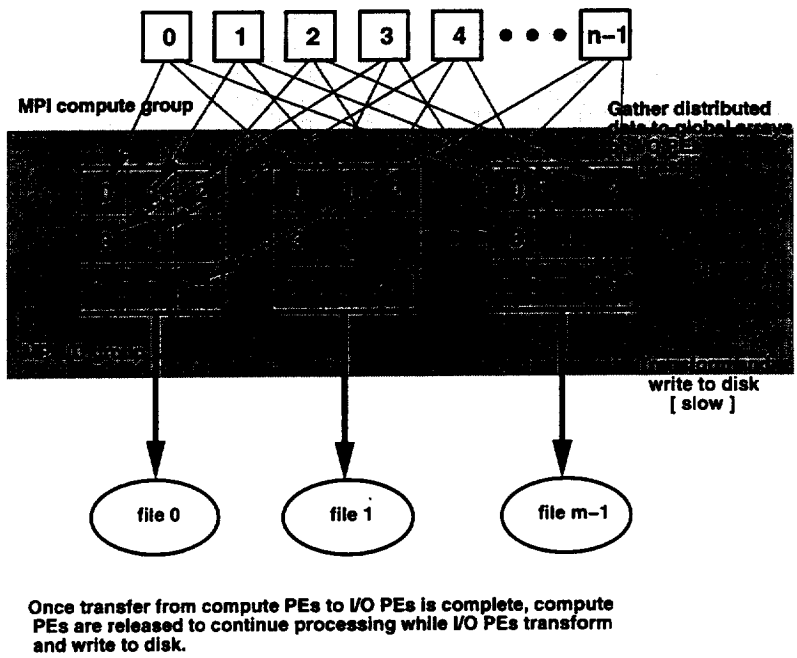


Figure 1: In the GPIOS parallel I/O concept, PEs are split into compute and I/O nodes. The top group of PEs is dedicated to the primary computation, while the middle row depicts a pool of I/O PEs. For output, each I/O PE writes to an independent file. While data input can be performed in an analogous manner, the amount of input data are small in comparison to the output data.

Parallel I/O Design

A conceptually simple way of writing output from a parallel application is to gather the data into global fields on a single processor which subsequently writes the data to a file. The I/O is serial and synchronous because all computation typically stalls until the I/O process is complete. If an application needs to process and write large amounts of data, this I/O time can significantly slow performance. For parallelizing the History module, we have adopted the conceptual simplicity of the single-node output model and modified it to allow performance gains. Rather than delegating I/O to a single PE that is also tasked with computation, we use a group of PEs that are dedicated to I/O (Figure 1). The compute PEs are delayed only long enough to gather the global fields on I/O PEs. The data transformations and physical disk I/O occur asynchronously on the I/O PEs while the compute PEs resume work on the GEOS DAS integration. The optimal number of I/O PEs is a function of the number of output streams, the number of compute PEs, and perhaps other factors such as the amount of memory on a single PE. This scheme is portable to any machine with an MPI implementation.

Status & Future Work

A prototype GCM written in Fortran 90 that uses MPI for communication is nearing completion [3]. Key to building the GCM was the development of a set of base utilities to support the data decomposition and to help isolate MPI calls from the application. [2] In addition to routines that define and create the decomposition in a general way, there are routines that facilitate the easy scattering of data from a global state to a distributed state and vice versa. The gather utilities are used extensively by the parallel I/O system and are the key infrastructure for its implementation.

We have developed the parallel History module and tested it in a unit test framework. We have integrated it with the nearly complete MPI GCM prototype. The full system with parallel I/O has been tested on SGI Origin, DEC Alpha, and Cray T3E. Initial results are encouraging and show that large amounts of data can be written with little delay to the GCM integration. Preliminary comparisons between the MPI GCM and the shared-memory GCM indicate that History is much faster in the MPI version. Unfortunately, realistic experiments are not yet possible as we await further development of the full MPI GCM, however the following timings from a unit test show good I/O performance and scaling as the number of streams is increased.

I/O PEs	1	2	3	4
Compute PEs	4	4	4	4
Data size (MB)	56	112	168	224
Computation Wait Time (s.)	2.2	3.5	5.2	6.5
Effective I/O Rate (MB/s.)	25.5	32.0	32.3	34.4
Total I/O Time (s.)	231	240	242	240

Table 2: The timing and throughput results on the SGI Origin2000 are listed for different numbers of I/O PEs and a fixed number of Compute PEs. The computational wait time is the time required for the Compute PEs to transfer output data to the I/O PEs before resuming the GCM integration. As the data size increases linearly with the number of I/O PEs, the I/O rate and the overall I/O time remain nearly constant. The increase in data size is accomplished by adding more streams, thus more I/O PEs for efficiency.

For the number of PEs we expect to use with the first version of the MPI GEOS DAS, we think the dedicated I/O group concept presented here will provide adequate performance. This concept may not scale to large numbers of compute PEs due to increasing communication latency costs, although the use of asynchronous communication should help hide latency. As portable parallel I/O interfaces such as MPI-IO become widely available and supported by vendors, we plan to evolve GPIOS to make use of these tools. Issues such as the overhead of grid transformations and the requirement to write data in

a modified HDF format (HDF-EOS) will have to be addressed at that time. At this time most work has been focused on the GCM due to it's large portion of the I/O load, but attention must be given to I/O optimization in PSAS. It is expected that dedicated I/O nodes can also be successfully used in PSAS.

References

[1] K. Ekers, J. Stobie, S. Schubert, D. Lamich, A. da Silva, L. Takacs, C. Pabon-Ortiz, A. Molod, R Govindaraju, S. Nebuda *File Specification for GEOS-3 Gridded Output*. NASA Data Assimilation Office, 1998

[2] W. Sawyer, L. Takacs, A. da Silva, P. Lyster. The Design of PILGRIM: A Parallel Library for Grid Manipulations in Earth Science Calculations. NASA Data Assimilation Office Note, 1998

[3] W. Sawyer, R. Lucchesi, P. M. Lyster, L. L. Takacs, J. Larson, A. Molod, S. Nebuda, C. Pabon-Ortiz *Parallelization Aspects of an Atmospheric General Circulation Model for Data Assimilation*. Proceedings of High Performance Computing '98

Author: Chris H.Q. Ding
National Energy Research Scientific Computing Center
Lawrence Berkeley National Laboratory
Building 50F
Berkeley, CA 94720
cding@nersc.gov

Co-author(s): Peter M. Lyster
Jay W. Lawson
Jing Guo
Arlindo da Silva

Parallel Atmospheric Data Assimilation

Atmospheric data such as temperature, moisture, winds, etc., collected by satellites and ground observation stations provide only partial information about the atmosphere. They are assimilated to numerical forecasts to provide a coherent, evolving state of the global atmosphere. The data analysis system, the Physical-space Statistical Analysis System (PSAS) developed at the Data Assimilation Office at NASA's Goddard Space Flight Center, requires computing resources far beyond the capabilities of even the state-of-the-art vector supercomputers. We describe an efficient and scalable implementation of the PSAS on distributed-memory massively parallel supercomputers such as Intel Paragon and Cray T3E; the implementations achieves superb performance as demonstrated by detailed performance analysis of systematic runs on up to 512 processors on Paragon, T3D and T3E. Consequently, the solution time is reduced to 24.6 seconds on 512-PE T3E from 5 hours on a single head of Cray C90 for a real problem of 80,000 observations, a 740-fold reduction of turn-around time. We will discuss the code structures and the modular programming approach used to separate the original codes from those for parallelization.

Implementation of a Parallel Kalman Filter for Stratospheric Chemical Tracer Assimilation

L.-P. Chang*
General Sciences Corporation (a subsidiary of
Science Applications International Corporation).

P. M. Lyster‡, R. Ménard†,S. E. Cohn
Data Assimilation Office,
NASA/Goddard Space Flight Center, Greenbelt, Maryland

Additional affiliations:
‡ Department of Meteorology,
University of Maryland College Park
‡ Joint Center for Earth System Science.
† Joint Center for Earth System Technology,
University of Maryland Baltimore County

May 20, 1998

* E-mail: dao.gsfc.nasa.gov, phone: +1 301 805 6998, Mail: Data Assimilation Office, Code 910.3, NASA Goddard Space Flight Center, Greenbelt, MD 20771, USA.

Abstract

A Kalman filter for the assimilation of long-lived atmospheric chemical constituents has been developed for two-dimensional transport models on isentropic surfaces over the globe. An important attribute of the Kalman filter is that it calculates error covariances of the constituent fields using the tracer dynamics. Consequently, the current Kalman-filter assimilation is a five-dimensional problem (*coordinates* of two points and *time*), and it can only be handled on computers with large memory and high floating point speed. In this paper, an implementation of the Kalman filter for distributed-memory, message-passing parallel computers is discussed. Two approaches were studied: an operator decomposition and a covariance decomposition. The latter was found to be more scalable than the former, and it possesses the property that the dynamical model does not need to be parallelized, which is of considerable practical advantage. This code is currently used to assimilate constituent data retrieved by limb sounders on the *Upper Atmosphere Research Satellite*. Tests of the code examined the variance transport and observability properties. Aspects of the parallel implementation, some timing results, and a brief discussion of the physical results will be presented.

1 Introduction

This paper extends an earlier paper by Lyster el al. (1997) on the implementation of a parallel Kalman filter used for atmospheric data assimilation at the Data Assimilation Office (DAO) of the NASA's Goddard Space Flight Center(GSFC). Even though the Kalman filter has been applied to meteorological problems for almost a decade, its full implementation in the context of 4-dimensional data assimilation is not yet possible on today's computers.

In order to develop a Kalman filter that has certain practical use, we focus our model problem on the assimilation of relatively long-lived trace chemical constituents in the middle atmosphere. This problem is also of interest to the earth science community. If we choose the potential temperature as the vertical coordinate, then the assumption of two-dimensionality becomes a good approximation in describing the transport dynamics. Thus, we have implemented a Kalman filter in spherical geometry on an isentropic surface in the stratosphere with either a medium (4° *lat* $\times$ 5° *lon*) or a high (2° *lat* $\times$ 2.5° *lon*) resolution. Real-data observations for the Kalman filter assimilation currently come from the NASA *Upper Atmosphere Research Satellite* (UARS) limb-sounding instruments that obtain retrievals of trace gases in the stratosphere. Analyzed winds from the global atmospheric data analysis system of the NASA/GSFC/DAO are used to drive the transport model of the Kalman filter to assimilate real data.

In the following sections, we briefly discuss the Kalman filter equations, the implementation strategies of the filter for parallel computers, timing information of the filter on two different distributed-memory computers, some numerical tests and experiments, and conclusions of our work.

2 Kalman Filter for constituent assimilation

If we neglect the diabatic effects, chemistry, vertical mixing and explicit subgrid-scale parameterization of mass flux, we may then express the trans-

port of the long-lived trace constituents on an isentropic surface as

$$\frac{\partial \mathbf{w}}{\partial t} + \mathbf{V}_\theta . \nabla_\theta \mathbf{w} = \mathbf{0}.$$

Here $\mathbf{w}$ represents the mixing ratio of the constituent, and $\mathbf{V}_\theta$ is the 2-D wind vector on the isentropic surface. In matrix-vector notation, a discrete version of this equation can be written as

$$\mathbf{w}_k^t = \mathbf{M}_{k-1}\, \mathbf{w}_{k-1}^t,$$

where $\mathbf{w}_k^t$ is an n-vector of constituent mixing ratio on a grid covering the isentropic surface, and the $n \times n$ matrix $\mathbf{M}_{k-1}$ denotes the action of the discrete dynamics from time t_{k-1} to time t_k.

A discussion of the Kalman filter in full scope can be found, for example, in the thesis by Cohn (1982). The Kalman filter algorithm essentially consists of two steps in matrix-vector notation:

• forecast step:

$$\begin{aligned} \mathbf{w}_k^f &= \mathbf{M}_{k-1}\, \mathbf{w}_{k-1}^a\,, \\ \mathbf{P}_k^f &= \mathbf{M}_{k-1}\, \mathbf{P}_{k-1}^a\, \mathbf{M}_{k-1}^T + \mathbf{Q}_k\,, \\ &= \mathbf{M}_{k-1}\, (\mathbf{M}_{k-1}\, \mathbf{P}_{k-1}^a)^T + \mathbf{Q}_k\,, \end{aligned}$$

• analysis step:

$$\begin{aligned} \mathbf{w}_k^a &= \mathbf{w}_k^f + \mathbf{K}_k (\mathbf{w}_k^o - \mathbf{H}_k\, \mathbf{w}_k^f)\,, \\ \mathbf{K}_k &= \mathbf{P}_k^f \mathbf{H}_k^T\, (\mathbf{H}_k\, \mathbf{P}_k^f\, \mathbf{H}_k^T + \mathbf{R}_k)^{-1}\,, \\ \mathbf{P}_k^a &= (\mathbf{I} - \mathbf{K}_k\, \mathbf{H}_k)\, \mathbf{P}_k^f\, (\mathbf{I} - \mathbf{K}_k\, \mathbf{H}_k)^T + \mathbf{K}_k\, \mathbf{R}_k\, \mathbf{K}_k^T\,. \end{aligned}$$

The last equation is referred to as the *Joseph form* of the error covariance equation. When the optimal Kalman gain $\mathbf{K}$ is used, it simplifies to the following *optimal form*

$$\mathbf{P}_k^a = (\mathbf{I} - \mathbf{K}_k\, \mathbf{H}_k)\, \mathbf{P}_k^f\,.$$

In these equations, we have used the following notations: $\mathbf{M}$ denotes the transport model matrix; $\mathbf{w}$ denotes the state vector; $\mathbf{P}$ denotes the error covariance matrix; $\mathbf{Q}$ denotes the model error covariance matrix; $\mathbf{K}$ denotes the (Kalman) gain matrix; $\mathbf{R}$ denotes the observation error matrix; $\mathbf{H}$ denotes the interpolation matrix; The subscript 'k' denotes the k-th time step; The superscript 'f' denotes the 'forecast' step; The superscript 'a' denotes the 'analysis' step; The superscript 'o' denotes the 'observed' quantity; The superscript 'T' denotes the 'transpose' of a matrix.

3. Implementation strategies for distributed-memory parallel computers

The style of programming adopted in coding the Kalman filter is to have the same compiled program run on many processors with each processor responsible for different parts of the distributed memory. This is referred to as Single Program with Multiple Data (SPMD). Earlier code development was done partially on the Touchstone Delta and mainly on the Intel Paragon computers at the California Institute of Technology (CalTech) with the Intel NX communications library used for message passing. About a year ago, we have implemented the MPI library and have since made runs on the Cray T3E at NASA/GSFC.

In what follows, we briefly describe the implementation strategies for the forecast step and the analysis step, respectively:

(a)forecast step:
The state vector $\mathbf{w}_k^f$ is evaluated first. The linear discrete dynamics $\mathbf{W}_k$ depends on the wind field but not on $\mathbf{w}_k^f$. We then evaluate the covariance matrix $\mathbf{P}_k^f$. The covariance computation involved in the Kalman filter is floating-point count- and memory-intensive, so it is important to distribute effectively the large $(n \times n)$ matrix $\mathbf{P}$. We have considered two implementation strategies, the operator- and the covariance-decompositions:

i) operator decomposition
The operation $\mathbf{MP}$ can be represented as $[\mathbf{MP}_1, \mathbf{MP}_2, \cdots, \mathbf{MP}_i, \cdots, \mathbf{MP}_n]$,

where $\mathbf{P}_i$ is the ith column of matrix $\mathbf{P}$. Each $\mathbf{P}_i$ is state-vector-like with the same structure as $\mathbf{w}$. The operator decomposition is based on a decomposition of the domain of the transport model $\mathbf{M}$. For the forecast of $\mathbf{w}$ or a column of $\mathbf{P}$, this is a classical domain decomposition algorithm. The schematic for operator decomposition for storing large $(n \times n)$ matrices and performing $\mathbf{M}(\mathbf{MP})^T$ is shown in Fig. 1a.

ii) covariance decomposition

In this case, the covariance matrix $\mathbf{P}$ is partitioned along rows so that whole columns are stored contiguously on each processor. The transport model operator such that $\mathbf{M}$ operates on whole columns of $\mathbf{P}$. On a message-passing computer with number of processors $N_p \gg 1$, load balancing must be attained for efficiency. The schematic for covariance decomposition for storing large $n \times n$ matrices and performing $\mathbf{M}(\mathbf{MP})^T$ s shown in Fig. 1b.

iii) comparison of the two decompositions

Comparison of the measured speedup curves using the Touchstone Delta and the Intel Paragon for the two approaches discussed above are shown in Figs. 2a and 2b. Examination of them shows that the latter scales much better than the former. It is well known that parallelization of different transport models is, by and large, no easy task, especially for the semi-Lagrangian ones. The covariance decomposition approach makes it unnecessary to parallelize the transport model, and this enables the choice of transport scheme on the basis of scientific merit alone. This is the reason why we favor the covariance decomposition as the default approach of the Kalman filter program.

(b)analysis step:

The gain matrix $\mathbf{K}$ is evaluated first in the analysis step. For bilinear interpolation, the $(p \times n)$ matrix $\mathbf{H}$ has only four non-zero elements in each row, so it is treated as an operator in the code. Since $\mathbf{P}^f$ is distributed, and $\mathbf{K}$ is reproduced identically on all processors, partial sums of $\mathbf{P}^f\mathbf{H}^t$ on individual processors are first obtained then globally summed over all processors to

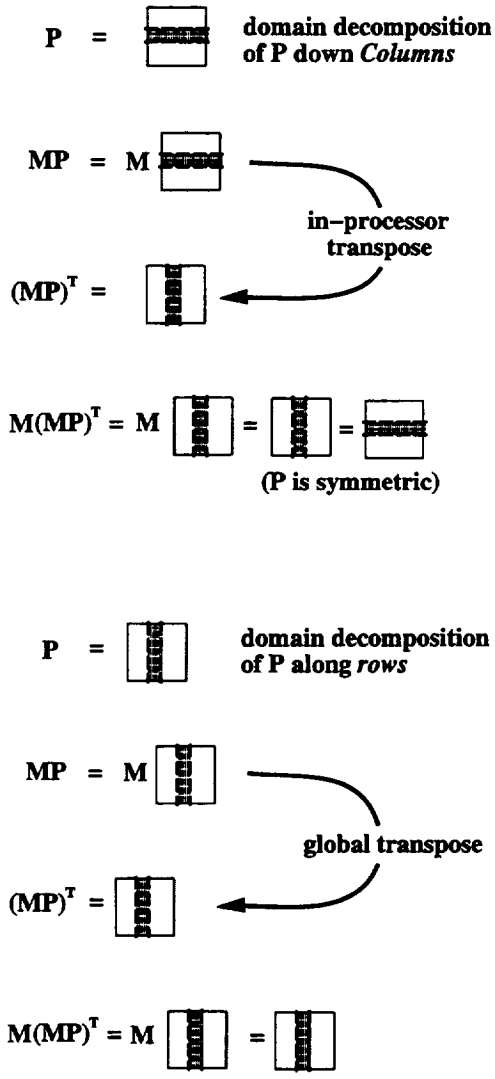


Figure 1: (a)Schematic for the operator decomposition approach; (b)Schematic for the covariance decomposition approach for storing large size-n^2 matrices and performing $\mathbf{M}(\mathbf{MP})^T$.

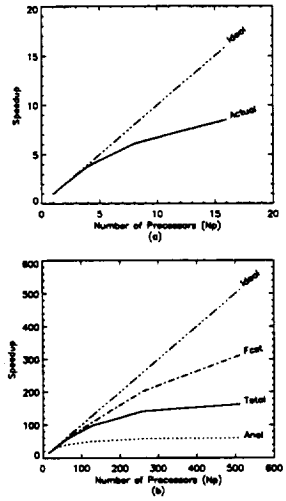


Figure 2: (a)Speedup curves for the domain decomposed van Leer transport algorithm implemented on the Touchstone Delta; (b)The actual speedup curves for the forecast step, the analysis step and the the full Kalman filter on the Intel Paragon for the medium resolution using covariance decomposition and the optimal form analysis equation.

get the net result. The matrix $\mathbf{HP}^f\mathbf{H}^T$ is evaluated as $\mathbf{H}(\mathbf{HP}^f\mathbf{H}^T)$, with the matrix $\mathbf{P}^f\mathbf{H}^T$ already existing on all processors. After $\mathbf{HP}^f\mathbf{H}^T + \mathbf{R}$ is obtained, its inverse is evaluated by an eigenvalue decomposition solver of a symmetric matrix. There is a memory burden in storing $\mathbf{K}$ and $\mathbf{P}^f\mathbf{H}^T$ on all processors, which becomes comparable to the storage of $\mathbf{P}$ when $p \approx n/N_p$.

After the evaluation of the gain matrix $\mathbf{K}$, we evaluate the covariance matrix $\mathbf{P}_k^a$. For the optimal form, $\mathbf{P}^a$ is evaluated as $\mathbf{P}^f - \mathbf{K}(\mathbf{HP}^f)$. The second term uses $\mathbf{K}$ and $(\mathbf{P}^f\mathbf{H}^T)^T$, both of which are identically stored on all processors. For the Joseph form, $\mathbf{P}^a$ is evaluated as $(\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)[\mathbf{P}^f - \mathbf{K}(\mathbf{HP}^f)]^T + \mathbf{KRK}^T$. which is generated from $\mathbf{HP}^f$, $\mathbf{K}$ and $\mathbf{R}$ all stored identically on all processors.

To finish up the analysis step, we evaluate the state vector $\mathbf{w}_k^a$. This is a relatively smaller calculation, and is carried out identically on all processors.

The innovation vector $\mathbf{w}^o - \mathbf{H}\mathbf{w}^f$ is evaluated and saved along with $\mathbf{w}_\mathbf{k}^\mathbf{a}$ for collection of innovation statistics.

4 Timings for the parallel Kalman filter

(a)with the Intel Paragon

We first examine the timing information for the Kalman filter using the Intel Paragon at CalTech. The speedup curves corresponding to the forecast step, the analysis step and the full filter for medium resolution have already been shown in Fig. 2b. Curves here using the Intel Paragon are obtained by directly connecting points corresponding to numbers of processors 8, 16, 32, 64, 128, 256 and 512. We chose these numbers partially because we had to comply with the queue arrangement at the time those runs were made. The actual times in seconds per time step for the analysis using the Joseph form, the forecast step and the full filter are shown in Fig. 3a for the medium resolution and p=14 observations per time step. The dominant cost of the analysis for large numbers of processors is clear. A typical 10-day assimilation run takes 960 time steps, which evaluates to 45 min. of wall clock time for the full filter with 256 processors. The corresponding result of the actual times in seconds per time step for the optimal form is shown is Fig. 3b. Since the optimal form is simpler arithmetically, the actual times for the analysis are relatively small. Only for large number of processors $N_p > 256$ does the time for the analysis step exceed that of the forecast step. A 10-day run for the optimal form takes about 34 min. of wall clock time of the full filter using 256 nodes due to the simpler calculation in the analysis step.

Due to the limitations of main memory, high-resolution runs can only be performed on 256 and 512 processors of the Intel Paragon. For a 10-day run with 960 time steps on 256 processors, the total time for the full optimal Kalman filter at high resolution is 7.6H. And it takes 5H using 512 nodes. The ratio of the total time for 256 to that for 512 processors is 1.52 which is considerably better than that for the medium resolution, 1.35.

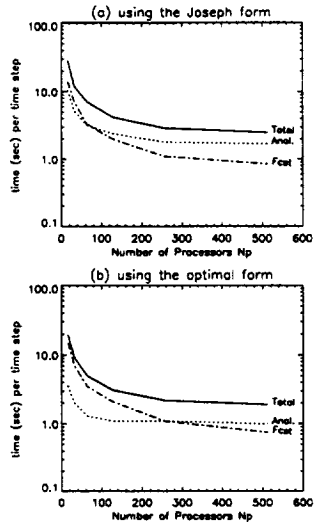


Figure 3: The actual time (s) per time step of the forecast step, the analysis step and the full Kalman filter on the Intel Paragon for medium resolution using covariance decomposition.

(b)with the Cray T3E

We will now show the timing information for the full filter using the Cray T3E at NASA/GSFC. Comparison of the medium-resolution speedup curves for the full filter for the Paragon and for the T3E is shown in Fig. 4a. In the Cray T3E case, the solid curve represents the connection of points for numbers of processors 10, 12, 16, 32, 46, 92, ..., 414, 460, 512 somewhat similar to the curve for the Paragon. Notice that multiples of the meridional dimension of the state vector, 46, were used in making runs. It is clear that the T3E is a faster computer than the Paragon by a factor of about 3. What is also shown in dashes is the actual step-function-like behavior of the Kalman filter performance as a result of the way the covariance decomposition is domain-decomposed in the code when the number of processors is greater than the meridional dimension of the state vector. Except for those numbers of processors mentioned above, additional numbers such as 91, 137, 183, ..., 459 were included in plotting the dashed curve. It is seen that use of number of processors from 47 up to 91 produces a nearly constant time per time step.

This implies two things: (1)the additional processors between 47 and 91 are not contributing to the speedup of the forecast step because fast processors have to wait for the slow ones to finish their work; (2)the speedup of the analysis step is not large enough to really improve the speedup of the total step. Use of 91 (=2*46-1) processors is the least economical choice.

For the high-resolution runs, the meridional dimension of the state vector is 91. The T3E has enough memory space to allow for runs with number of processors greater than or equal to 182. In analogy to Fig. 4a, we show the comparison of the full filter using, respectively, the Paragon and the T3E for the high-resolution grid in Fig. 4b. It is clear that the scalability for this resolution is markedly better than that for the medium resolution. It can also be inferred that the forecast step outweighs the analysis step roughly for all the numbers of processors used up to 512.

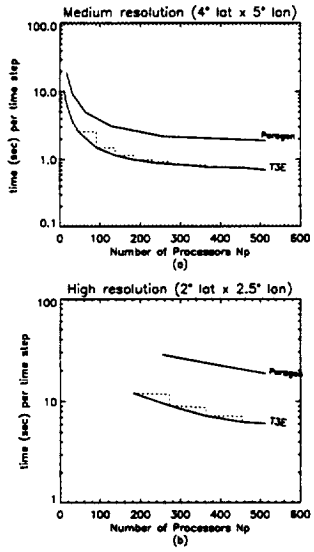


Figure 4: Comparison of the optimal-form timings of the full filter using, respectively, the Intel Paragon and the Cray T3E; Solid curves denote the envelopes , and the dashed curve illustrates the true behavior using the T3E.

5 Numerical tests and experiments

Results of two numerical tests using solid-body-rotation winds and two numerical experiments assimilating the mixing ratio of methane (CH_4) on the 1100K isentropic surface from two UARS limb sounders, the *Cryogenic Limb Array Etalon Spectrometer* (CLAES) and the *Halogen Occultation Experiment* (HALOE), will be briefly described below.

(a) test for consistent evolution of the error variance

For nondivergent flows with no observations, the variance is only being advected by winds. Use solid-body rotation with flow over the poles as the winds, we may test the implementation of the discrete covariance propagation. The variance field at time zero and that after a full rotation are shown in Fig. 5. Except for a slight north-south asymmetry, the overall shape is well preserved indicating a sound variance propagation near the poles. This test indicates that our covariance transport is correctly coded.

(b) observability test

This test involves both forecast and analysis steps using synthetic perfect observations to test reduction to zero of the total variance in finite time if the observability condition is met. Zonal solid-body rotation winds are used, and observations are made at all grid points along a fixed meridian at each time step such that the entire flow is observed perfectly in one rotation. Because the observation error variance matrix $\mathbf{R}$ is taken to be zero, the Joseph form of the analysis step is used to help ensure numerical stability in this extreme case. The initial error covariance is obtained from the SOAR covariance function with values of correlation length L=(1000, 500, 5km). The total variance versus time is plotted for each L value in Fig. 6. For L=1000 and 500km, where the correlation length is comparable to the grid spacing near the equator and greatly exceeds the grid spacing near the poles, the variance decreases rapidly at first, then decreases almost linearly, and finally reaches zero in one day. For L=5km, the correlation length is well below the grid spacing, corresponding to an almost diagonal initial covariance structure. In

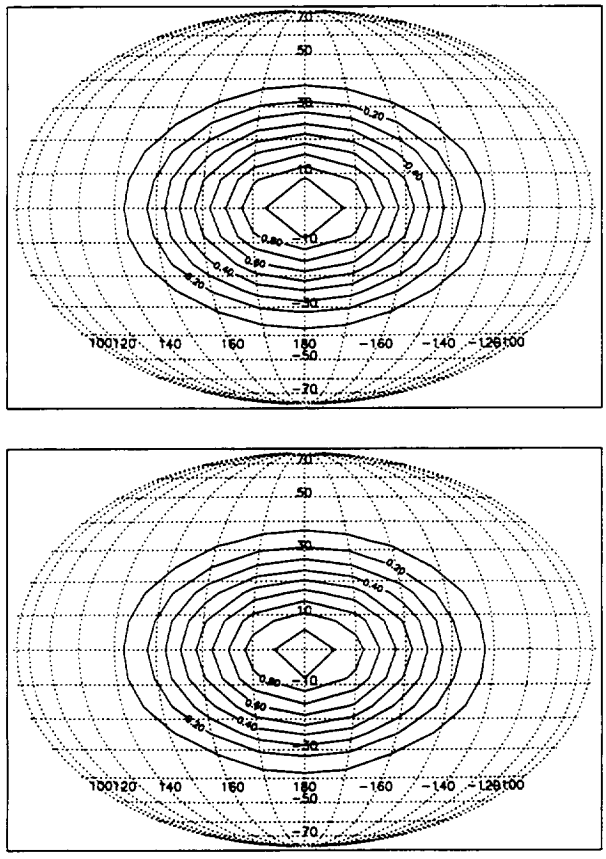


Figure 5: (a)The initial variance; (b)the final variance after a full rotation of the winds for solid-body wind propagation over the poles.

this case, the total variance is expected to decrease almost linearly with time, which is exactly what our test result shows.

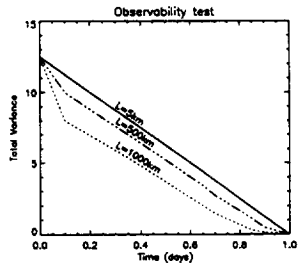


Figure 6: The total variance vs. time for a meridional observing network and an observation error covariance matrix $\mathbf{R}$=0. The initial error covariance matrix is obtained from SOAR covariance function with values of correlation length L= 1000, 500 and 5km. The rotation period of the solid-body winds about the polar axis is one day.

(c) UARS CLAES and HALOE experiments

A 4-day pure advection and two 4-day assimilations, respectively using the UARS CLAES (dense) and HALOE (sparse) observations, of the 1100K methane mixing ratio field have been conducted. Results for day-4 pure-transport and the CLAES assimilation are shown in Fig. 7. A wave breaking near the southern tip of Africa can be seen. The CLAES assimilation depicts the wave breaking pattern in finer detail when compared to the pure-transport result. The HALOE assimilation result shows a wave breaking pattern which resembles the CLAES result very much, even though only a few observations are available and are located in the tropics. This suggests that the covariance field in the Kalman filter helps to influence the wave breaking pattern in mid- latitudes in this case.

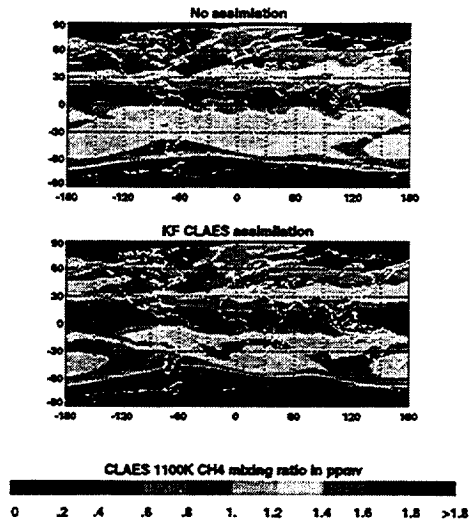


Figure 7: 1100K mixing ratio field for the pure transport and the Kalman filter assimilation of UARS CLAES observations on day 4.

The three-dimensional field of methane mixing ratio can be generated by a layering of results from a two-dimensional Kalman filter. Fig. 8 shows a sample 3-D plot.

6 Conclusions

Here we briefly summarize what we have accomplished in regard of implementing the parallel Kalman filter at DAO:

We have implemented on distributed-memory parallel computers a Kalman filter for the assimilation of atmospheric constituents on isentropic surfaces over the globe. The code has been parallelized using the Message Passing Interface (MPI), so it can run on any parallel computers that support this interface. We have thus far run this code on the Touchstone Delta, the Intel Paragon and the Cray T3D and T3E, all with success. Lately, many Fortran 90 features such as 'allocatable arrays', 'pointers to arrays' and 'modules' have been implemented in the most current version of the code.

We have developed a covariance decomposition approach as the basis of the

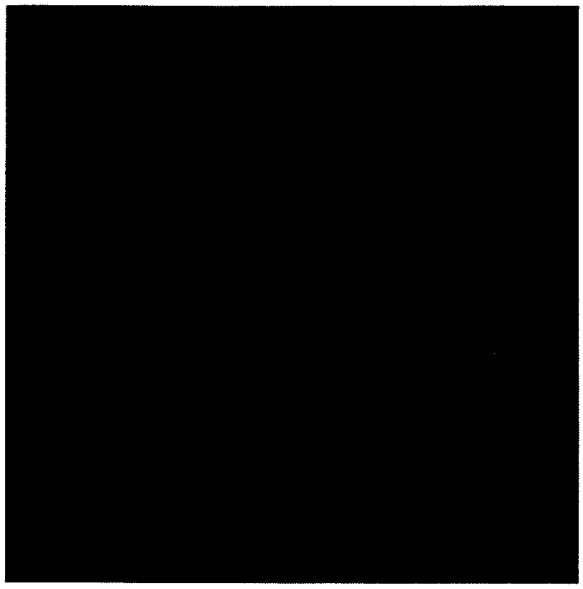


Figure 8: A sample plot of the three-dimensional field of methane mixing ratio generated by layering of results from a two-dimensional Kalman filter.

parallel algorithm in which we distribute the columns of the forecat-analysis error covariance matrix on different processors. This approach is not only efficient in terms of parallelization, it also has the important advantage that it is not necessary to parallelize the model transport code, only that it fits onto the memory of each processor, which is usually the case. The less scalable operator decomposition approach was abandoned because it also required parallelization of the transport model.

With regard to the wall clock speed, a 10-day run using UARS CLAES observations can be completed in 34 minutes for the optimal form of the analysis at medium resolution on 256 processors of the Intel Paragon with O4 and noieee compiler optimizations. The same run takes 13.6 minutes on the Cray T3E at NASA/GSFC, with some minor extra computation to improve the quality of the code. The T3E is about 2.5 times as fast as the Paragon at this resolution. For high resolution, a 10-day run using the

optimal form takes 5H on 512 processors of the Paragon and approximately 1.5H on the T3E. Note that at high resolution, the T3E is about 3.3 times as fast as the Paragon, which implies that our Kalman filter problem runs more efficiently on the T3E at high resolution. Another advantage in using the T3E is that its higher precision may help prevent potential numerical difficulties in the computation.

For medium resolution, the Kalman filter forecast step shows some reduction in scaling when the full 512 processors of the machines are used. This is primarily due to communication overhead involved in the global matrix transpose. The reduction in scaling for the analysis step is more severe due primarily to the serial (unparallelized) calculation of the Kalman gain matrix on each processor. This reduction is due, probably more significantly, to software simplifications that involve the use of global sum library subroutines. In contrast, for high resolution, the scaling seems to be reasonably good for up to 512 nodes.

A Comparison of several Scalable Programming Models

Alan J. Wallcraft
Naval Research Laboratory, Code 7323, Stennis Space Center, MS 39529.
wallcraf@ajax.nrlssc.navy.mil, +1 228 688-4813, Fax: +1 228 688-4759

Abstract

The NRL Layered Ocean Model (NLOM) is written in the tiled data parallel programming style, and uses an application specific programming interface to isolate operations that require communication. This allows different scalable programming models to be "plugged" into NLOM with relatively little effort. NLOM is similar to other OGCM's, except that it uses a direct Helmholtz's equation solver as part of its semi-implicit time scheme and typically runs with a very large horizontal extent and very few layers in the vertical. There are now several Fortran-based SPMD programming models to choose from on machines with a hardware global memory: a) MPI-1 message passing, b) MPI-2 put/get, c) BSP, d) SHMEM, e) F--, f) OpenMP, and g) HPF. These models are compared and contrasted based on actual experience with NLOM and related kernel benchmarks.

Introduction

The NRL Layered Ocean Model, NLOM, has been under continuous development for 20 years [1], [2], [3]. It has been used to model semi-enclosed seas, major ocean basins, and the global ocean. NLOM has been optimized for the problem space of Navy interest, simulation now-casting and prediction of fronts and eddies, and for such problems it is is 10-100 times more efficient (in operations performed per result) than competing OGCM's.

The current implementation of the model uses the tiled data parallel programming style. Consider the following simple serial code fragment:

```
      REAL A(IH+1,JH),DA(IH+1,JH)
      DO J= 1,JH; DO I= 1,IH
        DA(I,J) = DX*(A(I+1,J) - A(I,J))
      ENDDO; ENDDO
```

The arrays `A` and `DA` have been extended by a one column "halo" to allow a clean implementation of a periodic boundary. On entry `A(IH+1,:)` must be identical to `A(1,:)`. The equivalent tiled data parallel version adds a halo on all sides and splits the array into sub-domain tiles:

```
      REAL A(0:IHP+1,0:JHP+1,MP,NP),DA(0:IHP+1,0:JHP+1,MP,NP)
!HPF$ DISTRIBUTE A(*,*,BLOCK,BLOCK),DA(*,*,BLOCK,BLOCK)
      DO N= 1,NP; DO M= 1,MP
        DO J= 1,JHP; DO I= 1,IHP
          DA(I,J,M,N) = DX*(A(I+1,J,M,N) - A(I,J,M,N))
        ENDDO; ENDDO;
      ENDDO; ENDDO;
```

If MP and NP are both 1, this is Single Program Multiple Data (SPMD) domain decomposition. A 2-D, MPE by NPE, grid of processors are all running this identical program, with IHP=IH/MPE and JHP=JH/NPE. Provided the halo is up to date, the code fragment calculates the required values over the subdomain owned by the local processor. Alternatively, if MPxNP represents the number of processors, this is data parallel High Performance Fortran (HPF) [4] and the compiler does not need to generate any off-chip communication. It is also then appropriate for autotasking of the N loop using Fortran 77 compilers on SMP systems.

By using cpp macros, NLOM can select between scalable programming models at compile time while maintaining a single source code. An application specific programing interface (API) is used to isolate operations that require communication (halo updates etcetera). The API must be implemented for each new programming model, but the rest of the code is largely independent of the model used. For more information on scalable NLOM see Wallcraft and Moore [5], [6].

In the area of scalability, NLOM performs similarly to other OGCM's, except that it uses a direct 2-D Helmholtz's equation solver as part of its semi-implicit time scheme and typicaly runs with a very large horizontal extent and very few layers in the vertical. For example, a six layer 1/32 degree Pacific model is typical of "large" problems today and it has a 4096 by 2688 by 6 grid. Since it has so few layers in the vertical, NLOM uses 2-D domain decomposition (with the vertical dimension "on-chip") and performs all operations on 2-D slabs. OGCM's with more degree's of freedom in the vertical might still choose 2-D domain decomposition, but would typically perform communications on an entire 3-D field rather than on individual 2-D slabs. The direct 2-D Helmholtz's equation solver requires transposing from a 2-D to a 1-D domain decomposition, and therefore potentially reduce overall scalability. In general, scalability of NLOM is excellent on current scalable systems (using 64-256 nodes per job) because the 2-D arrays are so large.

SPMD programming models

There are now several Fortran-based SPMD programming models to choose from on machines with a hardware global memory.

MPI-1

Message passing is the most general scheme but it requires the source and target processor to cooperate in the transfer. MPI-1 is the message passing library of choise for SPMD codes, and is available on all platforms [7]. NLOM can use MPI and has cpp macros to hide word length differences and to select between several possible optimizations at compile time: (a) MPI_SENDRECV in place of the default non-blocking point to point calls, (b) SSEND in place of the default SEND, (c) replacements for ALLGATHER, ALLREDUCE(MAX) and ALLREDUCE(MIN) that use a binary tree on one dimension and a ring exchange on the other dimension, and (d) serialized array I/O.

SHMEM

SHMEM is Cray's one-sided put/get direct memory access library [8]. It is only suitable for machines with a hardware global shared memory. SHMEM is available on all Cray and SGI systems (Cray PVP, Cray T3E, SGI Origin 2000), but not on competing SMP or DSM systems from other vendors (e.g. Sun E10000 and HP/Convex SPP-2000). Unlike the other libraries described here, all SHMEM calls are (locally) blocking. Thus the standard Fortran assumption that there is a single thread of control and that any changes to memory or disk (buffers) caused by a subroutine call will happen before it returns is valid for SHMEM, but not necessarily for non-blocking calls in other libraries. The MPI-2 standard [9] has a good discussion of these issues, which can cause optimization problems in Fortran 77 but are much more serious for Fortran 90. SHMEM put updates memory on another processor, but this is not a problem if either (a) put is never used, or (b) the appropriate syncronization calls are included. The typical SHMEM program relies on a fast global barrier, and uses `COMMON` to hold arrays and/or buffers that are accessed from other processors. NLOM can use SHMEM and has cpp macros to hide word length differences and optionally to use local syncronization in place of some global barrier calls.

BSPlib

Bulk Synchronous Parallel delays put/get operations to the end of a "super-step", which allows implementation on machines without a global memory. Note that this implies that the put/get operations are non-blocking. There is a portable implementation, BSPlib, that runs on many machine types [10]. However, BSPlib effectively requires several global barriers at the end of each superstep because it imposes a particular order on puts and gets. There is formally no need for both put's and get's, and NLOM's SHMEM version (for example) never uses put, but there is no way to tell BSPlib to skip put processing. BSPlib has been designed to be called from C, e.g. sizes in bytes and byte offsets. There is a Fortran interface but it is a direct mapping of the C version, and is therefore very obscure to Fortran programmers. However, the library is small enough that it would be relatively easy to build your own (improved) Fortran interface. Unlike SHMEM, BSPlib only allows access to remote memory via pre-registered "windows". This potentially provides a safer interface, and allows non-static arrays to be accessed remotely, but at the cost of more complicated (and slower) code. BSPlib provides an alternative blocking get (on global shared memory machines only) that acts like a SHMEM get, and it is often possible to define a single memory window that includes all named `COMMON` areas. So BSPlib can be made to look almost exactly like SHMEM. However, BSPlib barrier performance prevents it being a viable (portable) alternative to SHMEM.

MPI-2

MPI-2 put/get is patterned on BSP, but with hooks that allow optimization for global memory machines (including non-global syncronization) [9]. If well implemented, this

will provide a portable alternative to SHMEM. MPI-2 includes all of MPI-1, and it also includes a very powerfull parallel I/O interface. Thus parts of MPI-2 are usefull even for message passing codes. It is also possible to use MPI-1 message passing for some things and MPI-2 put/get for others. However, there are currently very few MPI-2 implementations (none from US vendors). Like BSPlib, MPI-2 uses memory windows and non-blocking puts and gets. However, MPI-2's Fortran interface is much superior to that in BSPlib. As is typical of MPI, the MPI-2 one-sided interface is very rich. It is as easy to write a Bulk Synchronous Parallel program with MPI-2 as with BSPlib, but this involves using a very particular small subset of MPI-2's one-sided capabilities. It does not seem easy to "emulate" SHMEM using MPI-2, and such an emulation would certainly not be portable to all machines that might benefit from put/get. Fortunately, translating a SHMEM program to use (portable) MPI-2 should be straight forward. However, the performance of MPI-2 global barriers will be critical if it is to replace SHMEM. Some of the non-global syncronization options in MPI-2 may improve performance over global barriers, but fast global barriers are going to be essential if MPI-2 is going to gain wide acceptance by SHMEM programmers.

F--

F-- is a simple extension to Fortran that allows SHMEM-like put/get to be expressed via assign statements [11]. At a minimum this is a much clearer way to express put/get than a subroutine call. There are more concrete advantages, including lower latency (no subroutine call overhead) and the possibility of applying all the usual compiler optimizations to remote memory accesses. As a language F-- is currently incomplete because it cannot conform to Fortran I/O semantics but does not provide an alternative. There are experimental versions of F-- for the SGI Origin 2000 and the Cray T3E, but no compilers from other vendors. A major potential advantage of F-- over SHMEM (or MPI-2) is compiler optimization of fine grain code fragments involving remote memory accesses. However, this has yet to be demonstrated in practice. One problem area for optimization is that the compiler must assume that any variable marked for remote access could in fact be remotely accessed at any time during execution of that subroutine (variables only need be marked in subroutines that perform remote access). This has the effect of drastically reducing the optimization possibilites for such variables, so F-- could end up being slower than the equivalent SHMEM (or OpenMP) code. This could have been avoided by providing a more relaxed memory model as part of the F-- definition.

OpenMP

OpenMP is a set of compiler directives that provide a high level interface to threads in Fortran, with both thread-local and global memory [12]. OpenMP can also be used for loop-level directive based parallelization, but in SPMD-mode N threads are spawned as soon as the program starts and exist for the duration of the run. The threads act like processes (e.g. in MPI), except that some memory is shared and there is a single I/O name space. There are alternatives, but the closest mapping

to process-based SPMD programs is for almost all memory to be thread-local (i.e. one independent copy per thread) with global memory (visible to all threads) being used only as "buffers" for communication. A global buffer would typically hold N "local" buffers (one per thread). It is possible to use threads directly to create a threaded SPMD Fortran program, and portability is achievable via the Posix thread standard [13]. However, Posix threads are very low level and are difficult to use from Fortran. OpenMP provides a higher level, Fortran friendly, portable interface to threads. A threaded program has a single I/O space, and simultaneous calls from multiple threads may be unsafe. OpenMP has a more relaxed memory model than F--, that should not hinder optimization of shared variables.

HPF

High Performance Fortran provides a single-thread global memory user interface by doing communication and work distribution in the compiler, but it requires directives to distribute arrays across each processor's "local" memory [4].

Programming Issues

Portability

A language or library is "portable" if there are well understood guidelines for how to use (a subset of) the language or library so as to obtain good efficiency on a wide range machines (for a significant class of problems). SHMEM, F-- and OpenMP are unlikely to perform well on machines without a hardware global shared memory. BSPlib and MPI-2 put/get can take advantage of a hardware global shared memory, but can in principle also work on "shared nothing" systems, such as the IBM SP. How well MPI-2 will in fact work on such systems is unknown at present. A very low latency interconnect (and perhaps hardware support for barriers) might be all that is required to make MPI-2 put/get viable. Both HPF and MPI-1 can in principle be implemented efficiently on any scalable system.

MPI-1 is now available for all scalable systems, often via a vendor supported library. It is typically now possible to write a "portable" implementation of a given algorithm in MPI by following a few simple guidelines (defer syncronization, ISEND before IRECV, persistent communication requests, stride-1 buffers, don't use most collective operations). In addition, the syntax of MPI is regular enough that it is easy to provide several alternatives (selected at compile or run time). However, collective operations are often implemented very poorly. Thus a version using explicit point to point communications is almost always required for efficiency on some machines, with perhaps a MPI collective alternative for those few vendors who have optimized versions. Note that running many MPI collective operations twice on the same data is not guarenteed to produce the same result. This rules out such operations for many portable programs.

MPI-2 will probably become almost as widely available as MPI-1. It is not at all clear

today what will be required to write portable put/get code using MPI-2. The key unanswered question is how easy will machines, such as the HP/Convex SPP-2000, with two kinds of memory (local and global) be to program using MPI-2 put/get. A secondary portability concern is how efficiently vendors implement the various syncronization options. Since the efficiency of MPI-2 put/get may be low on at least some "shared nothing" systems, programs that must run on such machines would have to at least provide a MPI-1 message passing alternative to each put/get. This reduces the ease of use advantage for put/get over message passing. It is relatively easy to add MPI-2 put/get as an option to an existing MPI-1 message passing program (selectively replacing only those operations that are faster using put/get).

BSPlib is available as source code for many machine types and there is an effort underway to get vendor's to produce optimized versions. However, given that BSPlib is quite slow on machines with a global shared memory and MPI-2 can be used to write BSP programs, there does not seem to be much future for BSPlib as a portability tool.

HPF is widely available, but the language standard was not designed for portability. For example, there are no portable default array distributions so a portable program must include compiler directives in every subroutine defining the layout of every array used by that subroutine. It is also still the case that alternative distributions can produce huge differences in performance and (more importantly) that different distributions perform well with different compilers. One approach to HPF portability is to use the Portland Group HPF compiler, which is available on many platforms (i.e. use a portable compiler, rather than a portable source code).

SHMEM is a very small library providing very fast put/get. However, no vendor other than SGI/Cray has chosen to provide an implementation. A portable program that uses SHMEM today must provide an alternative (typically MPI-1) for non-SGI machines. For those looking to migrate SHMEM programs to an API that is portable across shared memory machines, the viable options seem to be MPI-2 and OpenMP. MPI-2 provides put/get but with significant differences to SHMEM and with unknown performance. OpenMP is available today with performance comparable to SHMEM, but migrating from SHMEM to OpenMP may require changes to subroutines that don't currently call SHMEM. The issue of I/O is particularly problematic.

There are experimental versions of F-- for the SGI Origin 2000 and the Cray T3E, but no compilers from other vendors. If other compilers existed, the major portability issue would be performance which at least initially might be relatively low because of the memory model required for global variables. How to implement F-- on machines with both local and global memory would also be an issue. F-- has the best syntax of all the alternatives for SPMD Fortran on global shared memory machines, but without a portable (source to source) compiler or support from several major vendors it is not a viable portability tool.

OpenMP is available in beta today from SGI on the Origin 2000, and from KAI as a source to source compiler on several machine types. It has wide support and should soon be available on all machines with a global shared memory, from PC's to MPP's. The standard is not rigerous enough to be confident about portability between the

many compilers that will exist. For example, it does not define the memory type (SHARED vs LOCAL) of variables with the SAVE atribute inside a subroutine. A program will definately break if a compiler allocates one kind of memory and the program assumes the other, so the only portable solution at present is to never use a SAVE statement in an OpenMP program (except for named COMMON). Once several implementations of OpenMP are available, it is likely that a portable subset of the language wil emerge. The only portable performance issue seems to be where global variables are placed in memory. OpenMP provides no mechanism to control this, and vendors are free to add their own (incompatable) extensions to OpenMP for laying out such arrays in memory. Some machines don't care about layout (e.g. Sun E10000) and some have run time layout mechanisms (e.g. SGI Origin 2000), but the performance on others may depend critically on shared array layout. Note that thread-local and shared variables map naturally to local and global memory respectively on machines with two kinds of memory. The only issue is where in global memory shared arrays are located.

Ease of Use

How easy each of the programming models is to use is obviously highly subjective. Message passing is certainly more difficult than put or get in that both sides of each memory transaction must cooperate in the exchange. This is more of an issue in cases with irregular communication patterns. The regular patterns typically associated with finite difference OCGM's are not usually difficult to express via message passing. The difficult part of put/get programming is syncronization, which is similar in all put/get models, but F-- is probably the easiest of all the process-based pure SPMD programming models to use.

A strong ease of use argument can be made for the global view of arrays provided by HPF. However, this is somewaht counter balanced by the difficulty of laying out arrays in memory. The extra boiler-plate code (compiler directives) needed for HPF programs is non-trivial. Many programmers seem to have "voted" for the less easy to use MPI-1, perhaps because HPF is easy to understand but does not necessarily provide a simple migration path from the existing code base. The performance of HPF relative to MPI-1 is also an issue.

OpenMP provides a programming model intermediate between F-- and HPF. It can use thread-local independent arrays, like F-- local arrays, or shared arrays, like HPF arrays, and can emulate F-- globally accessable local arrays using shared arrays with an extra dimension for the thread count. The primary difficulty with OpenMP is that SPMD threads that exist for the entire program are relatively new to Fortran programmers, and require some changes over process-based SPMD programming practices (particularly for I/O). Like all compiler directive based API's, the number of directives required can get out of hand (although it requires many fewer than HPF). OpenMP can be significantly easier to use than even F-- for irregular communications. For example a generic transpose operation in OpenMP might copy from one set of thread-local arrays (the input layout) into a shared array that uses the "nat-

ural" dimensioning and then copy out into a second set of thread-local arrays (the output layout). Both copy operations are trivial to program, and this works for any local distributions of the array. The real issue for OpenMP is not ease of use, but performance. In the transpose example, we have certainly done one extra copy of the entire array but this does not necessarily mean that this method is twice as slow as a direct copy from one layout to the other. In general, the fact that the programmer has no control over the layout of shared arrays in global memory might slow down some codes. However, threads are generally a big win over processes - particularly when mapping multiple threads or processes onto fewer processors.

I/O

Fortran has a specific model of I/O that is intrinsically single-thread, and which is violated by parallel I/O to a single file in all programming models except HPF. HPF can do parallel I/O that conforms to standard Fortran, but only if the compiler does this for you. All other API's except MPI-2 largely or completely ignore I/O. Generally serial writes from a single processor (or a single thread) works, as does parallel reads from any number of processors (but not from multiple threads). In some cases, parallel writes to non-overlapping records in a single file can be faster than serializing all writes - but there are no guarentees that this will work.

OpenMP has additional problems because there is just one process, and therefore one set of I/O files and pointers. Threaded I/O is actually well understood in C [13]. If the OpenMP Fortran's I/O library is "thread safe", any attempt to read and write in parallel to the same file (and perhaps to different files) will automatically be serialized. If the library is not safe, then the program must serialize I/O explicitly. Since there is only one I/O name space, only one thread should open and close a file and multiple reads of the same file from different threads will provide a different record to each thread. In contrast, for SPMD processes, each process must open and close a file it does I/O to and multiple reads of the same file from different processes will provide each with the same record.

NLOM inputs scalar control variables by reading them independently on all processors. This works well for process-based SPMD models, and is much less (programming) effort than the alternative of reading them on one processor and then broadcasting them to all others. This does not work for OpenMP, so NLOM now reads scalars into shared temporary variables from one thread under OpenMP (and into local temporary variables on all processes otherwise) and then copies the temporary variables into local variables on all threads/processes. This works with both threads and processes, but is not very transparent code. If OpenMP was the only target, it might be possible to leave input scalars in shared variables which would make the I/O code very similar to the uni-processor original (except for a few compiler directives).

MPI-2 contains an extensive API for parallel I/O. It is perhaps the most important reason for migrating from MPI-1 to MPI-2, particularly since the performance of MPI-2 put/get is as yet unknown. MPI-2 I/O looks like collective non-blocking message

passing. Very general patterns of I/O are allowed, but probably a much smaller subset will actually provide good performance. Portability is an issue, particularly since the API includes potentially machine specific "hints" on file layout etcetera.

The fact that MPI-2 I/O is non-blocking implies that it is asynchronous I/O. On typical scalable systems, with huge memory capacities, it is often practical to buffer an entire dump of all prognostic variables. Which suggests that most OGCM's really require asynchronous I/O more than they do parallel I/O. There is no standard method for specifying asynchronous I/O in Fortran, but if it is available OpenMP can easily implement asynchronous array I/O using a shared memory buffer (even though parallel I/O is not typically possible). Similarly a HPF compiler might provide non-standard asynchronous I/O. The other programming models may need sufficient unused memory on a single processor (rather than globally) to hold an entire dump of all prognostic variables before asynchronous I/O becomes a possibility.

Computation and Communication

In the interests of portability and flexibility, NLOM (like many other domain decomposition codes) separates computation and communication into distinct phases of the algorithm (and into distinct subroutines). However, there are cases where overlap of computation and communication is desirable or even essential. BSPlib and SHMEM do not allow such overlap at all. MPI-2 put/get is non-blocking, but may be implemented like BSPlib on some machines. There are MPI-1 non-blocking message passing calls, which certainly reduce overall latency when sending several messages but may not allow true overlap of communication and computation. In HPF, all communciation is scheduled by the compiler and overlap of communication and computation is one way for the compiler to achieve good performance but it is largely outside the programmers control. F-- does not allow overlap except at the level of the compiler's scheduling of loads and stores, but it does provide very low latency which may make algorithms with intermixed communication and computation viable (also true to a lesser extent for SHMEM and MPI-2 put/get). OpenMP has similar latency to F--, and threads provide the only guaranteed user-level method to control the overlap of communication and computation (one thread communicates while another computes). OpenMP SPMD threads are not the most suitable starting point for this kind of thread use, but they are probably still easier to use than native threads. A good example of latency hiding by using threads is SC-MICOM [14], which hides the communication cost between SMP "boxes" by having more sub-tiles than processors and doing the sub-tiles near the edge of the tile first and then updating, via MPI-1, the halos with the other SMP boxes while the interior sub-tiles are calculated. This is also an example of two level parallelization (threads and MPI-1), which is probably going to become more common. The combination of OpenMP and MPI-1 provides the most opertunity for latency hiding, but MPI-2 put/get for near communication and and MPI-1 message passing for far communication is probably also going to become very common.

Porting to NLOM

NLOM was originally designed so that the single source code worked for data parallel compilers (CM Fortran) and for SPMD message passing. In addition to replacing 2-D loops with 4-D loops (which can also help in cache reuse), this required 2,600 HPF DISTRIBUTE directives and 500 HPF INDEPENDENT directives. The directives are implemented via cpp macros, to allow for machine and compiler specific variations (e.g. CM Fortran and HPF). The total code is 69,000 lines of Fortran 77 including 22,000 standard comment lines of which 500 are compiler directives (many are repeats in different dialects). In addition there are another 60,000 lines of comments in a standard format required for all Navy operational models. The communication API consists of 32 subroutines, and 10,000 lines of code are used in total to implement the various versions (autotasking, data parallel, MPI-1, SHMEM). There are 6,500 lines of code in five versions of 16 machine specific (primarily I/O) routines, and there is also significant parallel programming model specific, and machine specific, code in the direct Helmholtz's equation solver. Overall the single node version of NLOM would actually use 41,000 lines of code including 15,000 comments.

Adding support for OpenMP required generating 6,500 lines of code for OpenMP alone, although most of these are identical to the SHMEM version. The shared parts of the code required 900 OpenMP compiler directives, 500 to characterize all COMMON's (could be reduced using INCLUDE) and 400 primarily to handle I/O. The I/O logic required other modifications, as outlined in the I/O section above, so that all I/O is performed by the master thread only. The OpenMP standard does not allow SAVE to be used for local variables in a portable program. NLOM already used named COMMON for most such variables, because of previous portability problems with local SAVE. However, local variables initialized with a DATA statement are implicitly saved and several of these had to be removed from NLOM to allow OpenMP to work.

Adding MPI-2 put/get will formally require modifications to 6,500 lines of code, but most of these will be identical to the SHMEM version. Only 110 SHMEM GET calls will need replacing, plus any necessay modifications to the synchronization logic. Additional macros will be required to allow some subroutines to use MPI-1 and others to use MPI-2 on a machine by machine basis.

Since NLOM already has an array I/O API that is called collectively by all nodes (9 subroutines, 700 non-comment lines), adding MPI-2 I/O should be straight forward. For example, adding support for the IBM "Parallel I/O File System" required only 50 additional lines of code.

Test problems

Three NLOM-based benchmarks are used to evaluate performance. Source code is available at **ftp://ftp7320.nrlssc.navy.mil/pub/wobnch**.

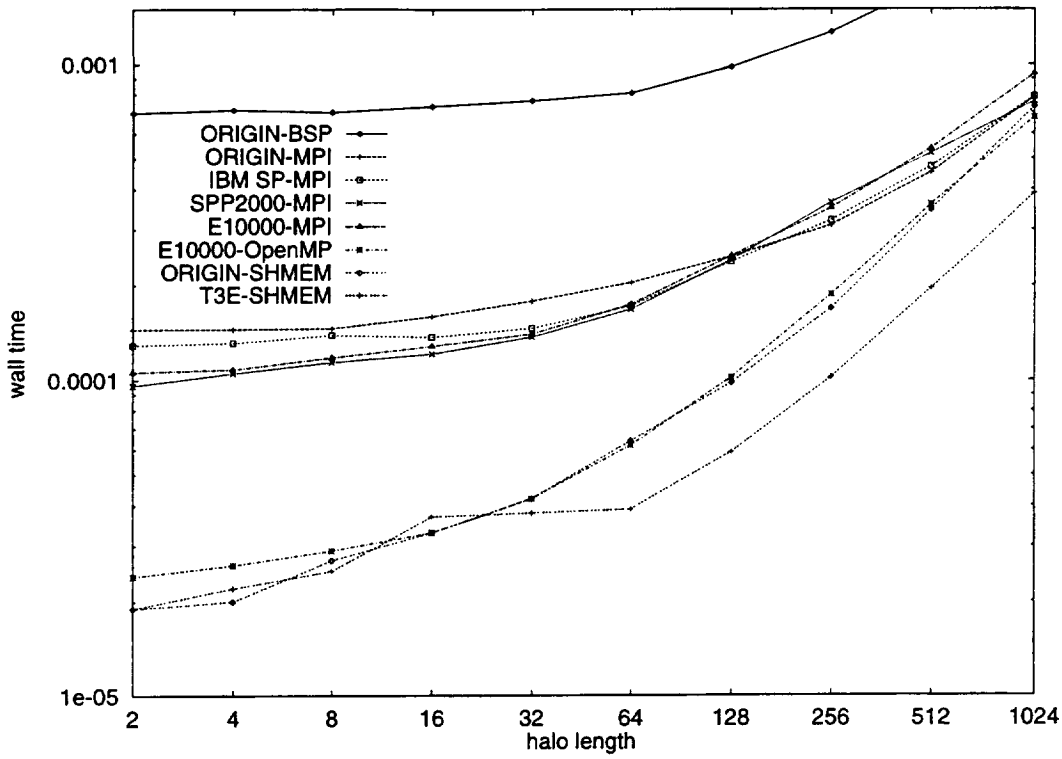


Figure 1: Best HALO times on 16 processors

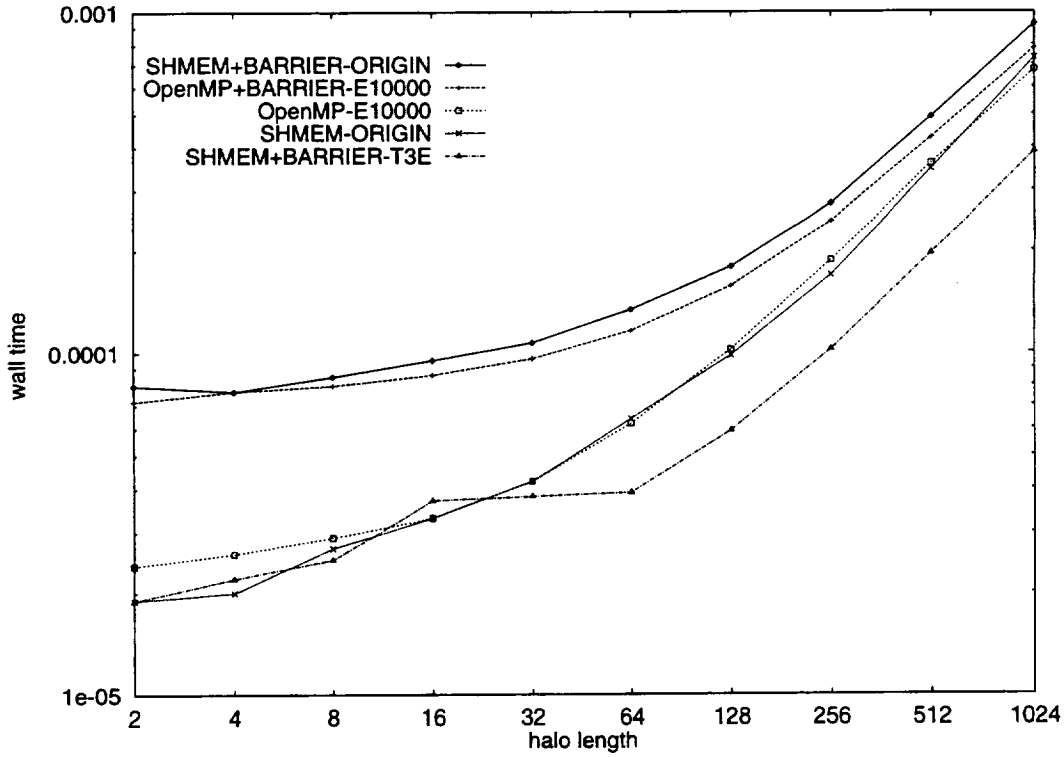


Figure 2: Shared memory HALO times on 16 processors

HALO

The HALO benchmark simulates a NLOM 2-D "halo" exchange for a N by N sub-domain with $N = 2...1024$. There are separate versions for each programming model. These can be used to compare exchange strategies for a given programming model, or to intercompare models. HALO puts a premium on low latency, but so does NLOM as a whole and HALO performance correlates well with overall NLOM communication performance. Figure 1 shows performance for the best HALO implementation of several programming model on a range of 16-processor machines. BSPlib is very slow, apparently because a "superstep" barrier involves three actual barriers. The best MPI-1 implementation is typically persistent ISEND then IRECV, and MPI-1 performance is similar on all scalable systems shown. Note that the "shared nothing" IBM SP does about as well as shared memory systems using MPI-1. Finally, the 1-sided memory methods are fastest (i.e. have the lowest latency) where applicable. Figure 2 shows 1-sided memory methods in more detail, and illustrates that local synchronization is faster than global barriers except on the Cray T3E.

RBSOR

machine	library	nodes	RBSOR	XCTILR	XCNORM	speedup
Cray T3E	SHMEM	16	4.902	0.100	0.782	(450 MHz)
Cray T3E	SHMEM	32	2.035	0.077	0.414	2.41 x16
Cray T3E	SHMEM	64	1.115	0.067	0.233	1.83 x32
Cray T3E	SHMEM	128	0.580	0.046	0.123	1.92 x64
SGI Origin 2000	SHMEM	16	3.908	0.969	0.769	(195 MHz)
SGI Origin 2000	SHMEM	28	1.687	0.308	0.366	2.32 x16
SGI Origin 2000	SHMEM	56	0.924	0.199	0.218	1.83 x28
SGI Origin 2000	OpenMP	16	2.697	0.156	0.549	(195 MHz)
SGI Origin 2000	OpenMP	28	1.540	0.109	0.477	1.75 x16
SGI Origin 2000	OpenMP	56	1.061	0.285	0.299	1.45 x28
Sun E10000	Sun MPI	16	8.940	1.883	1.489	(250 MHz)
Sun E10000	Sun MPI	32	3.873	1.166	0.915	2.31 x16
Sun E10000	Sun MPI	56	1.793	0.504	0.501	2.16 x32
HP SPP-2000	HP MPI	16	3.401	0.486	0.651	(180 MHz)
HP SPP-2000	HP MPI	32	1.614	0.212	0.356	2.11 x16
HP SPP-2000	HP MPI	64	0.761	0.153	0.214	2.12 x32
IBM SP	IBM MPI	16	2.580	0.227	0.625	(135 MHz)
IBM SP	IBM MPI	32	1.562	0.204	0.465	1.65 x16
IBM SP	IBM MPI	64	0.955	0.163	0.324	1.64 x32
IBM SP	IBM MPI	128	0.892	0.167	0.411	0.98 x64

Table 1: Time in seconds for 27 2048x1344 Red-Black SOR solves

The RBSOR benchmark is a stand alone test of the red-black SOR iterative solver

used by NLOM. Three wall clock times are recorded, a) total (RBSOR), b) halo exchange (XCTILR), and c) global sum (XCNORM). This benchmark is much simpler to get running than the full NLOM code, and it provides some indication of both computation and communication perforamance on a given machine. However, the computational kernal of RBSOR is not necessarily representative of NLOM as a whole (compare table 1, RBSOR, with table 2, NA824). The OpenMP times on a SGI Origin 2000 compare favorably with SHMEM times. The Sun E10000 is showing super-scalar speedup, but relatively poor computational kernal speed.

NA824

machine	method	nodes	time	Mflop/s	speedup
Cray T3E-900	SHMEM	14	44.1 mins	1,064	(450 MHz)
Cray T3E-900	SHMEM	28	21.0 mins	2,236	2.10x 14 nodes
Cray T3E-900	SHMEM	56	10.2 mins	4,591	2.06x 28 nodes
Cray T3E-900	SHMEM	112	5.7 mins	8,184	1.79x 56 nodes
Cray T3E-900	SHMEM	224	3.4 mins	13,601	1.68x112 nodes
SGI Origin 2000	SHMEM	14	75.3 mins	622	(195 MHz)
SGI Origin 2000	SHMEM	28	31.7 mins	1,481	2.38x 14 nodes
SGI Origin 2000	SHMEM	56	15.5 mins	3,031	2.05x 28 nodes
SGI Origin 2000	SHMEM	112	7.8 mins	6,030	1.99x 56 nodes
SGI Origin 2000	OpenMP	14	96.9 mins	484	(195 MHz)
SGI Origin 2000	OpenMP	28	38.0 mins	1,233	2.55x 14 nodes
SGI Origin 2000	OpenMP	56	21.1 mins	2,225	1.80x 28 nodes
SGI Origin 2000	OpenMP	112	12.7 mins	3,682	1.65x 56 nodes
HP SPP-2000	MPI	14	56.3 mins	833	(180 MHz)
HP SPP-2000	MPI	28	25.1 mins	1,868	2.24x 14 nodes
HP SPP-2000	MPI	56	15.1 mins	3,107	1.66x 28 nodes
IBM SP	MPI	14	39.2 mins	1,197	(135 MHz)
IBM SP	MPI	28	20.0 mins	2,345	1.96x 14 nodes
IBM SP	MPI	56	11.2 mins	4,169	1.79x 28 nodes
IBM SP	MPI	112	7.7 mins	6,060	1.45x 56 nodes
IBM SP	MPI	224	5.1 mins	9,208	1.51x112 nodes

Table 2: Performance of NLOM (NA824)

The NA824 benchmark is for 3.05 model days on a 1/32 degree 5-layer Atlantic Subtropical Gyre region (grid size 2048 x 1344 x 5). The run includes all the typical I/O and data sampling, but it does not measure initialization time (before the first time step). The sustained Mflops estimate is based on a hardware trace of a single processor Origin 2000 run (without MADD ops), i.e. is "useful" flops only. Like most heavily used benchmarks, this is for a problem smaller than those now typically run. The NA824 speedup from 28 to 56 processors is similar to the 112 to 224 speedup for

the four times larger 1/64 degree Atlantic model. Illustrating that NLOM is indeed a "scalable" code. Table 2 summarizes the performance results. Note that for 28 processors and above 1/8th of the tiles are being discarded at compile time because they are over land, thus the 28 processor wall time is equivalant to a 32 processor wall time with no discarded tiles. Linear speedup from 14 to 28 processors is not 28/14 but 32/14 (i.e. not 2x but 2.29x). The Cray T3E is showing the best scalability to large numbers of nodes, but the IBM SP is competative on up to 64 processors. The SGI Origin 2000 is showing a sustained cache effect, with speedups of two or more for each doubling of nodes. OpenMP on the Origin is currently slower than SHMEM, but communication routines perform similarly between the two methods. So OpenMP compilation is slowing down the compuational kernals. This is a beta compiler and improvements can be expected in the future. The HP/Convex SPP-2000 is faster then the SGI Origin 2000 if only about half of the 16 processors in each hypernode are used (the 14 and 28 processor runs were on 2 and 4 hypernodes respectively). Like many other SMP systems, the SPP-2000's memory bandwidth does not sustain all the supplied processors when running memory bound jobs.

Conclusions

Retrofitting a scalable programming model to an existing scalable ocean code such as NLOM is not an ideal basis for comparision, even though NLOM is designed to accept alternative programming models. The separation of communication and computation phases for much of NLOM, and the fitting of each programming model into the existing communication API, puts at a disadvantage programming models that are easy to use and that favor mixing of communication and computation. Even so, this comparison provides a baseline for performance on an actual application. Early OpenMP compilers are showing promise, but MPI-2 put/get will probably be most programmers first exposure to 1-sided communication. We must hope that MPI-2 implementations will approach the performance of SHMEM and OpenMP.

Acknowledgements

This is a contribution to the 6.2 Global Ocean Prediction System Modeling Task. Sponsored by the Office of Naval Research under Program Element 62435N. Also to the Common HPC Software Support Initiative project Scalable Ocean Models with Domain Decomposition and Parallel Model Components. Sponsored by the DoD High Performance Computing Modernization Office. The benchmark simulations were performed under the Department of Defense High Performance Computing initiative, on (i) a SGI Origin 2000 and a HP SPP-2000 at the Naval Research Laboratory, Washington D.C., (ii) a Cray T3E at the Naval Oceanographic Office, Stennis Space Center, Mississippi, and (iii) an IBM SP at Waterways Experiment Station, Vicksburg, Mississippi.

References

[1] Hurlburt, H.E. & J.D. Thompson (1980). A numerical study of Loop Current intrusions and eddy shedding. J.Phys. Oceanogr., **10**, *pp 1611-1651.*

[2] Wallcraft A.J (1991). The NRL Layered Ocean Model users guide. NOARL Report **35**. *Naval Research Laboratory, Stennis Space Center, MS, 21 pp.* **http://www7300.nrlssc.navy.mil/html/images/users_guide.ps.gz**

[3] Moore D.R. & A.J. Wallcraft (1996). Formulation of the NRL Layered Ocean Model in Spherical Coordinates. NRL Contractor Report **CR 7323–96-0006** *Naval Research Laboratory, Stennis Space Center, MS.*

[4] Koelbel C.H., D.B. Loveman, R.S. Schreiber, G.L. Steele Jr., M.E. Zosel (1994). The High Performance Fortran handbook. MIT Press.

[5] Wallcraft A.J & D.R. Moore (1996). A Scalable Implementation of the NRL Layered Ocean Model. NRL Contractor Report **CR 7323–96-0005** *Naval Research Laboratory, Stennis Space Center, MS.*

[6] Wallcraft A.J & D.R. Moore (1997). The NRL Layered Ocean Model. Parallel Computing **23**, *pp 2227-2242.*

[7] Snir M., S.W. Otto, S. Huss-Lederman, D.W. Walker, J. Dongarra (1996). MPI: the complete reference. MIT Press.

[8] Cray Research Inc. (1996) Application Programmer's Library Reference Manual. Cray Research SR-2165

[9] Message Passing Interface Forum (1997) MPI-2: Extensions to the Message Passing Interface. **http://www.mpi-forum.org/docs/docs.html**

[10] Goudreau, M.W., J.M.D. Hill, K. Lang, B. McColl, S.B. Rao, D.C. Stefanescu, T. Suel, T. Tsantilas (1996) A proposal for the BSP worldwide standard library. **http://www.bsp-worldwide.org**

[11] Numrich R.W., J.L. Steidel, B.H. Johnson, B.D, de Dinechin, G. Elsesser, G. Fischer, T. MacGonald (1997) Definition of the F-- *extension to Fortran 90. Proc. 10th Int. Workshop on Language and Compilers for Parallel Computers Springer-Verlag*

[12] OpenMP Organization (1997) OpenMP Fortran Application Programming Interface **http://www.openmp.org**

[13] Kleiman, S., D. Shah, B. Smaalders (1996) Programming with threads SunSoft Press. Prentice Hall.

[14] A. Sawdey, M. O'Keefe, W. Jones. A General Programming Model for Developing Scalable Ocean Circulation Applications. 1996 ECMWF Workshop on the Use of Parallel Processors in Meteorology **http://www-mount.ee.umn.edu/ okeefe/micom/**

Issues in the Design of Parallel Ocean Circulation Models

Aaron C. Sawdey
Cray Research
Eagan, Minnesota
sawdey@cray.com

Matthew T. O'Keefe
University of Minnesota
Minneapolis, Minnesota 55455
+1 612 625-6306
okeefe@ece.umn.edu
http://www-mount.ee.umn.edu/~okeefe/micom

Abstract: In this extended abstract we describe a programming model for parallel ocean circulation codes. We have applied this model to the Miami Isopycnic Coordinate Ocean Model (MICOM) and are in the process of applying it to the Princeton Ocean Model (POM). The model exploits highly parallel machines that have memory and network hierarchies to achieve scalable, efficient performance combined with ease of programming.

Introduction

Since the early 1980's, the growth of the computing speed of the largest supercomputers has been dramatically increased through the use of parallel processing. As a result of this increase in computing power, many fields have replaced physical experiments and tests with computer simulation and modeling. This is certainly true in ocean circulation and in climate research. However, it can be difficult to develop applications that run efficiently on supercomputers with hundreds or thousands of processors.

Several approaches have been taken to the difficult task of writing parallel programs. From the programmer's perspective, the easiest method might be to use a parallel language such as *High Performance Fortran* (HPF) that supports array parallelism. In parallel languages like HPF, the programmer has a global view of the data being manipulated and the compiler assumes responsibility for distributing the data and work across a parallel machine. Shared-memory parallel programming using compiler directives such as the new *OpenMP* standard is becoming more widely used for smaller numbers of processors. However, direct shared-memory programming relies on efficient shared-memory support from the underlying hardware. Thus it is not directly applicable to clustered machines or very large systems such as the Cray T3E and IBM SP/2 that do not have the necessary level of shared-memory support. The other common method of creating parallel programs is to write a message-passing program. With this method, the programmer is required to write a program from the perspective of a processor in a parallel machine, explicitly specifying when the processor should send or receive messages to and from its neighbors. If written correctly, message-passing parallel programs are efficient and achieve high performance; however, they can also be difficult to write and debug.

Standards such as HPF have created languages that can be used to write parallel programs. Unfortunately, these languages often demand too much of the compiler. Because of the amount of information a HPF compiler must discover through analysis to produce an efficient parallel program, the compilers end up either producing low execution performance or being very complex. Compiler complexity can lead to incomplete implementation of features deemed less important by the compiler developers, larger numbers of compiler bugs, and excessively long compile times.

An easier solution that works for many problem areas is for the programmer to write codes that conform to a specific coding style. This enables us to apply straightforward compiler analysis to automate some of the difficult tasks required to produce a parallel program. In addition, we can make some useful assumptions about the code because of the guidelines the programmer has agreed to follow. These assumptions would be difficult to validate in an arbitrary program, even if advanced compiler analysis techniques were used. It is also possible that a programmer unaware of these assumptions would inadvertently violate them. We have used a specific coding style we call *self-similar programming* to accomplish this; this style is particularly applicable to applications in ocean circulation.

Self-Similar Programming

Our goal is to make it easier for programmers to write parallel programs that have properties that will enable them to run well on parallel supercomputers:

- The fraction of the program that must be executed by a single processor or thread of control should be very small.
- The communication patterns of the program should map well to the interconnection network of the underlying hardware.
- The parallel work should be as coarse-grained as possible; that is, there should be as much work between synchronizations as possible.

We accomplish this goal by providing the programmer with two complementary pieces. The first piece is the self-similar programming model, which we believe leads to programs that have all of the properties listed above . The second piece is a powerful data-flow analysis tool designed to help the programmer with the problems that arise in writing self-similar parallel programs [4].

Our approach is a middle ground between writing programs in a new parallel language like HPF and applying a sophisticated parallelizing compiler to existing code. Both of these approaches are more general than ours but each also has its limitations. For example, some programs (including some self-similar programs) cannot be fully analyzed by parallelizing compilers because they use programming techniques that can only be analyzed with the use of data available when the program is running. However, a good parallelizing compiler should be be able to produce efficient code from a self-similar parallel program if it can avoid excessive synchronization between parallel loop nests. Unfortunately, it appears that current commercial parallelizing compilers are not able to avoid synchronizing after every parallel loop nest.

Our programming model applies to programs that have the *self-similarity* property, though not all dimensions or axes of the program are required to have this property. The program dimensions that are self-similar have the following characteristics:

- Loops along the dimension span the entire length of the dimension. In other words, if a dimension of the arrays in the program is declared to contain elements *1* to *N*, then loops that index that dimension should run from *1* to *N* as well.

- When the program is computing new values for index i of a dimension, it may use other elements from indices $i+d$ of that dimension. The value of $|d|$ must have an upper bound b such that $b << N$, where N is the length of the dimension. Because of this characteristic, data use along each self-similar dimension must be local.

- The elements along the dimension can be computed in any order. There are no inter-iteration dependences in loops along the dimension.

Each dimension of the program which is self-similar is called a *parallel dimension*, and the dimensions that are not self-similar are called *serial dimensions*. Loops that index a parallel dimension are called *parallel loops*. The loops that do not index a parallel dimension are called *non-parallel* or *serial loops*. A self-similar code can be represented as a mathematical operator F that produces a new state at point p from the old values at p and neighbors of p:

$$p_{t=i+1} = F(p_{t=i}, neighbors(p_{t=i}))$$

A more general definition of a self-similar algorithm is that the same algorithm used for updating the entire problem domain can be used to update any subsection of the domain without altering anything other than loop bounds, array bounds, and boundary conditions. This is the origin of the term in this context; the algorithm for a large domain is the same as the algorithm for a small domain, which is analogous to the way fractal structures have the same characteristics when observed at a variety of different length scales. Both these definitions hold for many ocean circulation codes, which use regular, finite-difference numerical techniques in the horizontal (parallel) dimensions. The vertical dimension is most often serial.

Applications

The Miami Isopycnic-Coordinate Ocean Model (MICOM) [1] has been our primary test case for self-similar parallel programming using overlap areas. MICOM models the circulation of water in the ocean. This circulation is driven by density differences within the ocean and by interaction with the atmosphere at the surface. Evaporation, which is affected by water temperature and air humidity, and precipitation determine whether there is a net loss or gain of water at the surface. Wind both adds horizontal momentum to the surface waters and adds turbulence, which affects the depth to which water is mixed vertically. The difference between water and air temperature also affects vertical mixing. When the air is colder than the water, it cools the water and causes vertical mixing due to convection. Except for the vertical mixing that takes place near the surface, there is very little vertical mixing within the ocean. Water masses with different densities remain segregated and do not mix.

In order to capture this segregation of water by density within the ocean, MICOM treats the ocean as a stack of variable-thickness layers. The topmost layer, known as the *mixed layer*, has a variable density and represents the layer that is vertically mixed due to its interaction with the atmosphere. The layers below the mixed layer are assigned fixed densities, which increase with increasing depth. The term isopycnic used in the name of the model refers to this constant-density treatment.

Because the heat capacity of the ocean is several orders of magnitude larger than that of the atmosphere, accurate heat transport within the ocean is important for long-term climate simulations. The density-based vertical coordinate used in MICOM is one way to achieve this by avoiding artificial mixing of water with different densities within the ocean model.

The equation of state of salt water, which determines density as a function of temperature and level of salinity, can be solved for temperature as a function of salinity and density. This allows MICOM to store only the salinity of the water at each point of each internal ocean layer; when temperature is desired, it can be computed from the equation of state given the density and salinity. The densities of the internal layers of ocean in MICOM are fixed parameters; they are set by the user at the beginning of a simulation.

MICOM uses finite-difference approximations to solve differential equations describing the motion of water within each horizontal layer. Two sizes of time step are used to time-integrate MICOM's state. The main time step is used to integrate the equations describing the behavior of each element of each of the horizontal layers. However, barotropic gravity waves, which affect all of the layers, propagate many times faster than any other signal in the model. To improve computational efficiency, this barotropic signal is subtracted out, propagated for many small time steps with a single-layer model, and then added back into the multi-layer model. The ratio between the barotropic timestep and the main timestep is approximately *20:1*.

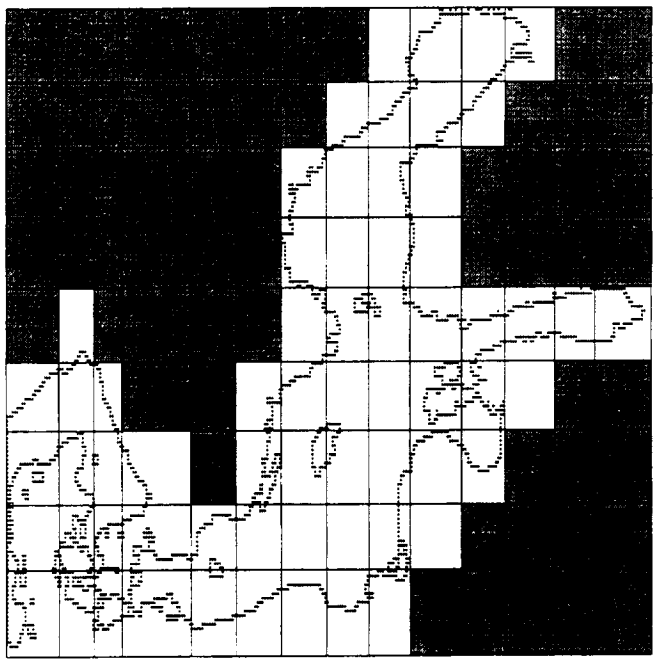


Figure 1: Domain decomposition of the Baltic Sea with land subdomains removed.

In Figure 1 we show a domain decomposition over the Baltic Sea. Note that subdomains in the partitioning that contain no water (dark grey in the figure) are not included in the parallel calculation, improving efficiency and reducing memory [3].

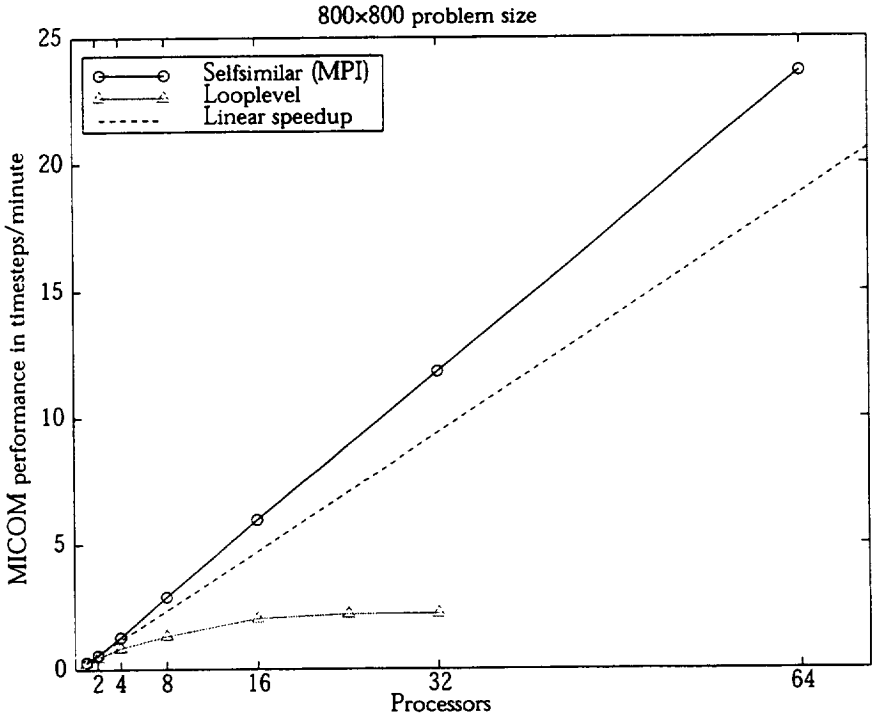


Figure 2: The performance of the parallel MICOM and MICOM 2.6 using loop-level parallelism in timesteps per minute for an 800x800 problem size on an SGI Origin2000.

In Figure 2 we show the scaling achieved for the parallel self-similar MICOM. This code was parallelized manually but later analyzed using our compiler tool TOPAZ [4] to determine the extent of the overlap regions required for parallel execution. The self-similar parallel MICOM scales super-linearly to 64 processor on the Origin2000. In contrast, the loop-level parallel code generated automatically using the SGI optimizing parallel Fortran compiler has limited scalability past 4 processors. This is due primarily to the order-of-magnitude increase in the number of barrier synchronizations required and in the reduced locality of this code.

Our current work includes applying our compiler tool TOPAZ to analyze and parallelize the Princeton Ocean Model (POM) [2]. We intend to reuse much of the parallel code infrastructure constructed for the MICOM effort. In addition we continue to perform scaling and performance testing of the MICOM code on highly parallel MPPs, DSMs, and clusters to better understand limits to its performance and scalability.

References

[1] R. Bleck and E. Chassignet, "Simulating the Oceanic Circulation with Isopycnic-Coordinate Ocean Models," in S. Majumdar and E. Miller, editors, *The Oceans: Physical-Chemical Dynamics and Human Impact,* chapter 2, pages 17-39, Pennsylvannia Acad. Of Science, 1994.

[2] A. Blumberg and G. Mellor, "A Description of a Three-Dimensional Coastal Ocean Circulation Model," in N. Heaps, editor, *Three-Dimensional Coastal Ocean Models,* vol. 4, pg. 208, American Geophysical Union, 1987.

[3] Aaron C. Sawdey, Matthew T. O'Keefe, Rainer Bleck, and Robert W. Numrich, "The Design, Implementation, and Performance of a Parallel Ocean Circulation Model", *Proceedings of the Sixth ECMWF Workshop on the Use of Parallel Processors in Meteorology*, Reading, England, November 1994. Proceedings published by *World Scientific Publishers* (Singapore) in **Coming of Age**, edited by G-R. Hoffman and N. Kreitz, 1995.

[4] Aaron Sawdey and Matthew O'Keefe, "Program Analysis of Overlap Area Usage in Self-Similar Parallel Programs," to appear in *Proceedings of the 10th International Workshop on Languages and Compilers for Parallel Computing,* Minneapolis, MN, August 1997.

Author: John M. Levesque
Applied Parallel Research
1723 Professional Drive
Sacramento, CA 95825
levesque@apri.com

Optimizing POP for a Cache Based Architecture

Today the major computational resources for the scientific community are cache based processors with high megahertz rates and multiple functions units, their Macho flop numbers approach a Gigaflop for a single processor. When researchers port their application to these systems they see anywhere from 4 to 100 MFLOPS. The cause for this lower flop rate is the utilization of the cache on the microprocessor. Given a very high megahertz rate the memory cannot keep up with the processor and the cache is placed between memory and the CPU to provide higher memory bandwidth. This high memory bandwidth is only achieved when all of the cache line fetched to the cache is used.

The Parallel Ocean Program (POP) was developed for the CM5 and MPP's in Fortran 90. When first ported to the SGI - Origin it achieved a maximum of 30-40 MFLOPS on a single processor. Over the past six months the performance of POP has been increased to over 100 MFLOPS per node. In order to achieve this increase in performance cache utilization was increased to the point that there are no cache misses other than those from the first fetch of a line into cache.

In the process of optimizing POP we developed a strategy and used tools that can be applied to other applications. The approach was to map out the computational arrays in the cache for the major looping structures. The layout of the initial POP program resulted in significant level 1 and 2 cache misses due to the two way associative cache. We found that re-mapping the arrays to eliminate these misses was facilitated by moving the depth dimension of the arrays to the first index. This also necessitated that the K (depth) loop become the inner loop for the computation. The cache mapping was performed using a cache analysis tool named CacheVU.

The study found that the F90 array syntax was not conducive to cache re-use and that the F90 program used many temporary arrays that hurt cache utilization. The solution to this problem, given that the POP developer desired to program in F90 array syntax was to translate the F90 to F77 using a pre-processor called zAPR. The details of the zAPR translation were formulated during the POP analysis. Cooperation between the tool developers and the POP developers has resulted in a F90 programming style that can be efficiently translated into optimal code.

This paper is the result of a collaboration between Christopher Kerr, Bob Malone and John Levesque.

PERFORMANCE OF BAROTROPIC OCEAN MODELS ON SHARED AND DISTRIBUTED MEMORY COMPUTERS

S. Piacsek and A. Wallcraft
Naval Research Laboratory, Code 7320
Stennis Space Center, MS 39529-5004, USA
piacsek@nrlssc.navy.mil, +1 228-688-5316, FAX: +1 228 688-4759
P. Jayakumar
University of Southern Mississippi/NAVO-MSRC PET Program
Stennis Space Center, MS 39529-5004, USA
L. Lonergan
Northrup Grumman Corporation/NAVO-MSRC PET Program
Stennis Space Center, MS 39529-5004, USA
M. Young
Naval Research Laboratory, Code 5593
Washington, DC 20375, USA

Abstract

The efficiency of explicit time integration schemes for barotropic models of the Mediterranean were investigated, in context of the vectorization and parallel modeling approaches employed on different architectures. The main focus of interest was the scalability and MFlops output of the codes as a function of domain size.

For simulations with real winds, mesh sizes ranged from 25 km down to 1.8 km (grids of 180x64 to 2048x1024), with the coarse resolution resolving only major straits like that of Sicily, and the high resolution even narrow straits like Gibraltar and Messina. Since the memory requirement of these grids only ranged to 70 Mbytes, we also performed simulations with idealized, precomputed winds for which mesh sizes ranged down to 280m, to produce a total memory requirement of 4 Gbytes. The analysis and interpretation of the latter results for the Mediterranean has not been performed yet. The explicit scheme consisted of the leap-frog scheme for the Coriolis, pressure gradient and advection terms, and 'lagged' times for the diffusion terms. The platforms utilized included the CM500-E (with CMF), the Cray C90 and T90 (with FT90 -O3 auto-tasking), the Cray T3E (with HPF and MPI), the SGI Origin2000 (with f77 -pfa -O2 power fortran,HPF and MPI), the IBM SP2 (with HPF) and the Sun Global Works (with HPF).

The MPI version of this code employed a 2-D tiling decomposition, and parallel runs were performed up to 512 processors on the T3E and up to 64 processors on the SGI Origin. The T3E 512 processors achieved an 82 % scaling efficiency relative to 32 CPU's. The SGI 64 processors achieved a scaling efficiency of 100 % vs. 32 processors, but less than linear for smaller number of processors. The auto-tasking versions were quite efficient even for small program sizes (17 Mb) and for small number of processors, with the SGI -pfa compiler option (with -O2 optimization) giving scalings of 1.9, 3.7, and 15.4 for

2, 4 and 16 CPU's, respectively, while the Cray T90 -O3 option (with FT90) gave scalings of 3.6 and 6.6 for 4 and 8 processors, respectively.

indent MFlops output reached 11.7 GFlops for the 512 node T3E, and 7.4 GFlops for the 128 node O2K.

1 Introduction

The recent advances in high-performance computing, especially on massively- parallel machines, have encompassed ocean models as well. These advances also include the reformulation and design of numerical schemes especially suited for parallel machines. On the one hand, progress has been achieved by porting older codes to new architectures; on the other hand, some codes have been designed from scratch for the new machines. In the 1992-1998 period at the Naval Research Laboratory, great advances were made in ocean modeling on a series of parallel computers. The principal machine used up-to-now by the group was the CM5-E, with 256 nodes and a total memory of 4 Gigawords; currently they are the Cray T3E, with 512 nodes and a total memory of 128 GB, and the SGI Origin2000, with 128 processors and 32 GB of memory.

The large memory and throughput of these machines enabled us to push the frontiers of ocean modeling much further, solving some problems important for Navy environmental prediction and climate simulation. Among these were the simulation of the reapparance of the 1983 El Nino 8 years later in the western Pacific, having traced the westward propagation of the constituent Rossby waves with sufficient spatial resolution and phase accuracy [Jacobs et al,1994]. The crucial factor in ocean modeling has always been resolution, both in the horizontal and vertical. High resolution reduces truncation and dispersion errors; in addition, it allows the size of the friction coefficient to be kept small, increasing the amplitudes of the ocean currents. A further advantage is the better resolution of gradients and the decrease in eddy sizes that can be resolved, extending the solution spectrum and energy cascade, crucial for long term simulations. A large part of the lateral friction employed in ocean models is necessary for numerical rather than physical reasons, to keep solutions stable and smooth.

There appeared to be two problems in the beginning of our parallel computations that had to be tackled: (a) how to migrate existing ocean models from a shared memory to a distributed memory computer; (b) a rethinking of the numerical techniques and algorithmic approaches to take advantage of the massively parallel nature of the new computers. On data-parallel machines, such as the CM5, this migration required a complete rewrite of the existing model codes into CM Fortran (CMF). On distributed memory machines, the use of HPF (High Performance Fortran) was facilitated by the existing CMF codes, from which HPF codes could be generated in about a week or two. Some of the large 3-D codes were ported directly from the Cray C90 to the T3D and T3E via message-passing: this typically involved inserting MPI message-passing function calls into regular f77 subroutines. On massively parallel machines, such things as communicating between neighboring grid points are major

issues, and the problem revolves around a trade-off between keeping all processors busy ("load balancing") versus keeping communication down between processors as much as possible ("locality management"). Furthermore, certain numerical schemes that rely on recursion relations (e.g. the well-known tridiagonal algorithm) encounter the same problems of conflict and inefficiency on parallel machines as they do on vector machines. Thus alternate formulations for inverting tridiagonal matrices (e.g. the Buzbee-Golub cyclic reduction technique [Buzbee et al,1971; Schwarztrauber,1977]) had to be invoked.

For the major part of the ocean modeling subroutines, it was relatively easy to rewrite the existing codes; however, certain Helmholtz solvers required major recoding efforts to arrive at an efficient program representation (Wallcraft and Moore, 1997). One problem we found with parallel computers is that performance on certain codes was degraded by necessary actions that are required at only a small number of grid points (e.g. the computation of boundary conditions), since all other processors are idle while these actions are performed.

For a review of some recent efforts in parallelizing ocean models, the reader is referred to Smith et al (1992), Dukowicz et al (1993), Piacsek and Wallcraft (1993), Bleck et al (1995), Webb (1995), Oberhuber and Ketelsen (1995), Wallcraft and Moore (1997) and Ma et al (1998).

The present report will focus on barotropic ocean models that employ explicit time integration, and do not require the use of a Helmholtz solver. In Section 1 we introduce barotropic models and give will a rationale for their use. In Section 2 we will give model details, including the numerical scheme for the explicit time stepping version, and some sample results in the Mediterranean. In Section 3 we present some performance figures on various parallel platforms.

2 Barotropic Models

2.1 Utility of Barotropic Models

Barotropic ocean models are 2-D and represent the ocean with one deformable layer, obtained upon integrating vertically the 3-D equations of a hydrostatic ocean. They include topography of the ocean bottom and (generally) a uniform density. All 3-D ocean models contain a barotropic mode (i.e. the vertically averaged motion). For a discussion of these topics, the readers are referred to Bryan,1969,1979; Madala and Piacsek,1977; Blumberg and Mellor,1987; Wallcraft,1991 and Dukowicz and Smith,1994.

It may be asked why barotropic modelling is done at all except in conjunction with baroclinic modelling? We can give three reasons immediately:

(a) in certain oceanic regions, especially in the winter season, deep convective mixing can occur which will tend to homogenize the density field over most of the depth range, so that the barotropic part may represent a significant if not the dominant component of the total circulation. Thus we can gain a useful insight into its patterns by using only a barotropic model, at a much lower computational cost.

(b) tidal forces create the greatest response in the barotropic mode, and their response must be modeled in shallow waters.

(c) The third, and probably most important, point is that the free surface elevation couples directly to the barotropic mode. The associated surface gravity waves, with their fast propagation velocities, can cause great problems for the numerical computations. For various numerical and computational reasons, this makes the solution of the barotropic equations the most CPU intensive, and hence expensive. Hence it becomes very cost-effective to study the efficiency and accuracy of the numerical techniques on parallel platforms in the 2-D setting of barotropic models, rather than in a costly 3-D baroclinic setting.

(d) The central role of the free surface in barotropic models becomes even more important when altimeter measurements of the sea surface elevation are used as part of the initialization/updating procedure for real-time prediction. This is because the information about the free surface elavation is first passed through the barotropic mode before its pressure effects are felt by the whole water column. Thus barotropic models are used to estimate the atmospheric-pressure induced sea surface elevations, not only the simple inverse barometer effect, but the so-called 'non-isostatic' response due to moving weather patterns. Both this 'non-isostatic' response, and the surface elevations due to tidal forcing, are needed to calibrate altimeter measurements of sea surface height [Kantha, 1995].

2.2 Model Equations for a Barotropic Ocean

To shorten and simplify the numerical description, we will present only the finite difference form of the relevant equations in Cartesian coordinates, using constant horizontal friction coefficients; extension to spherical geometry and variable friction coefficients is straightforward. We further assume a constant density, and that the ocean depth is much greater than the free surface elevation, so that only a linear form of the continuity equation needs to be utilized. In the same vein, because of the smallnes of barotropic currents in deep water, we omit the nonlinear advection terms in this description, though they were included in the code and the simulations.

We will use the nonlinear bottom friction formulation (6), and locate the variables on a staggered mesh called the Arakawa C-grid [Wallcaft,1991]. In this arrangement the pressure p and height h variables are located at the center of the mesh boxes, and the mass tansports U and V at the center of the box boundaries facing the x and y directions, respectively.

$$\begin{aligned}\frac{U_{ij}^{n+1}-U_{ij}^{n-1}}{\Delta t} &= fV_{ij}^{n}-\frac{gH}{\Delta x}(h_{i+1/2j}-h_{i-1/2j})^{n} \\ &+\frac{A}{\Delta x^{2}}(U_{i+1j}+U_{i-1j}+U_{ij+1}+U_{ij-1}-4U_{ij})^{n-1} \\ &+(\tau_{x})_{ij}-C_{d}.U_{ij}^{n-1} \qquad (1)\end{aligned}$$

$$\frac{V_{ij}^{n+1} - V_{ij}^{n-1}}{\Delta t} = -fU_{ij}^{n} - \frac{gH}{\Delta y}(h_{ij+1/2} - h_{ij-1/2})^{n} + \frac{A}{\Delta y^2}(V_{i+1j} + V_{i-1j} + V_{ij+1} + V_{ij-1} - 4V_{i,j})^{n-1} + (\tau_y)_{ij} - C_d.V_{ij}^{n-1} \quad (2)$$

$$\frac{h_{ij}^{n+1} - h_{ij}^{n-1}}{\Delta t} = -\frac{(U_{i+1/2j} - U_{i-1/2j})^{n}}{\Delta x} - \frac{(V_{ij+1/2} - V_{ij-1/2})^{n}}{\Delta y} \quad (3)$$

where

$\vec{U} = (U, V)$ - mass transports in the x- and y-directions, respectively

$\vec{\tau_w} = [(\tau_w)_x, (\tau_w)_y]$ - wind stress components

$\vec{\tau_b} = [(\tau_b)_x, (\tau_b)_y]$ - bottom stress components

h - free surface elevation

$f = 2\Omega \sin\theta$ - Coriolis parameter for latitude θ

$H(x, y)$ - topography (ocean depth)

A - coefficient of lateral friction

The bottom stress τ_b can be related in a linear or nonlinear way to the bottom velocity, which in this case has to be replaced by the depth-mean averaged velocity.

The use of a lateral friction coefficient is a general requirement for modeling all hydrodynamic processes that have strong nonlinearities, and as such it becomes a necessary part of ocean models as well. Such friction is also necessary for physical reasons, to represent the subgrid-scale mixing processes.

We assume closed boundaries for our rectangular domain, for which the relevant conditions are U = V = 0. Note that the vanishing of the depth H on land precludes having to specify the gradients of h, and no loss of parallel efficiency will occur.

2.3 Explicit Time Integration

Explicit time integration schemes are attractive because they do not involve matrix inversion or the use of iterative solvers. Though we avoid having to use a matrix inverter to solve for the values at t^{n+1}, we pay the price by being restricted in the time step we can take. The size of the time step one can march with is governed by the well-known Courant-Friedrichs-Levy (CFL) stability condition. For wave equations the time step is limited by the wave speed, in this case the speed of the surface gravity waves c_w, and is given by

$$\Delta t < \Delta x / c_w \quad (4)$$

Typically, $c_w = \sqrt[2]{gH}$ of the gravity waves exceeds 200m/sec in basins of depth $> 4000m$, so Δt is of order 60 sec for a spatial resolution of 14 km normally associated with eddy-resolving basin-scale (1/8 deg) ocean models.

We must also give expressions for the bottom stress τ_b in terms of the velocity components. In the non-linear approach, the bottom stress takes the form

$$\tau_{bx} = C_d.|U|U, \quad \tau_{by} = C_d.|U|V \tag{5}$$

with the value of the drag coefficient C_d taken to be either .0025 or .0050, depending on the author.

2.4 Barotropic Results in the Mediterranean

We have found that for this simple 2-D explicit code there were no impediments to either vectorization or parallelization. Our first parallel experiments were on the CM5. After the basic conversion from f77 to CMF, including calls to MAXVAL, SUM, etc., an attempt was made to speed up the code by studying serializing or parallelizing the different spatial dimensions, the size of these dimensions, and the handling of the boundary conditions. These have been reported on in Piacsek and Wallcraft (1993).

Using the CM5 code, we carried out simulations of the wind-driven circulation in the Mediterranean, including the non-isostatic response to moving atmospheric pressure gradients. For simulations with real winds on all platforms, mesh sizes ranged from 25 km down to 1.8 km (grids of 180x64 to 2048x1024), with the coarse resolution resolving only major straits like that of Sicily, and the high resolution even narrow straits like Gibraltar and Messina. Since the parallel versions of the wind interpolation routines have not yet been installed, and 6-hourly forcing at the very high resolution presented a forbidding amount of data transfer and storage, we ran the higher resolution cases only with analytic winds.

The model grid for the experiments presented here was 1024x512, giving a horizontal grid size of 3.5 km. The experiments were carried out on the 256-cpu CM500-E at NRL-DC, as well as on the various partitions of the CM5 at Minnesota (it ran even on the 64-cpu partitions). The horizontal diffusivity A was taken to be 50 m^2/sec. The model was forced with GCM-derived synoptic winds obtained from central weather prediction sites, and run typically for 60 days to equilibrium.

Figure 1 shows the transport vectors for the barotropic circulation in the Tyrrhenian Sea for the month of November 1994, The development of a strong cyclonic gyre in the southern half of the basin as winter approaches is quite evident.

Figure 2 shows the transport vectors for the barotropic circulation in the NE corner of the Eastern Mediterranean Basin for the four seasons of 1994. The strong effect of the topography, the so-called 'topographic steering' of the barotropic currents, is quite evident. The well-known, intense cyclonic gyre near the island of Rhodos (the 'Rhodes gyre') is well represented with only the barotropic mode, as is the westward

moving Anatolian current south of Turkey. The main seasonal changes appear to be the appearance of an anticyclonic gyre east of the Rhodes Gyre, and the tendency of the Rhodes Gyre to become an asymmetric bi-polar vortex pair, in the winter months.

3 MPI Version of Barotropic Code

The MPI version of this code employed a 2-D tiling decomposition, and parallel runs were performed up to 512 processors on the T3E and up to 64 processors on the SGI Origin. Since the memory requirement of the grids for the GCM wind simulations ranged to 70 Mbytes, we also performed simulations with idealized, precomputed winds for which mesh sizes ranged down to 280m, to produce a total memory requirement of 4 Gbytes. The analysis and interpretation of the latter results for the Mediterranean has not been performed yet.

In our initial phase of code development, we have used blocking send and receive calls. All processors were sending messages in parallel, but there was no overlap with any computations. The code sections shown below are not complete, citing only the statements relevant to illustrating the MPI approach. In the same vein, the number of processors and mesh dimensions are only illustration.

3.1 MPI Initialization

```
      PROGRAM BTPROGRAM_MPI
      include "mpif.h"
C
      parameter(nprocx = 8,nprocy=8)
      PARAMETER  (NX=2049,NY=1025)
      parameter(MyNX=((NX-2)/nprocx)+2)
      parameter(MyNY=((NY-2)/nprocy)+2)
C
      common /n1/ comm_2d, coords, left, right, above, down
      common /n2/ strided
C
      integer rank, size, ierr
      integer comm_2d, coords(2), left, right, above, down
      integer strided
      logical shift_up, shift_down, shift_left, shift_right
C
```

3.2 The Forecast Routine for the X-Transport U

```
      SUBROUTINE FCST_UVH
C
      integer left,right,above,down, coords(2),comm_2d,strided
      integer nprocx, nprocy,IS,JS
```

```
      logical shift_up, shift_down, shift_left, shift_right
      common /n1/ comm_2d, coords, left, right, above, down
      common /n2/ strided

      shift_up = .true.
      shift_down = .true.
      shift_left = .true.
      shift_right = .true.

            JS = 2
      if(coords(2) .eq. 0)  JS = 1
            IS = 2
      if(coords(1) .eq. 0)  IS = 1
      DO 100 J = JS,JJ1
      DO 100 I = IS,II1
      UU(I,J)   = U(I,J)
      VV(I,J)   = V(I,J)
      UVEL(I,J) = ZU(I,J)/HT(I,J)
      VVEL(I,J) = ZV(I,J)/HT(I,J)
  100 CONTINUE
C
      call shift_data(ZH,mynx,myny,
     & .false., .false., shift_left, .false.)
      call shift_data(UU,mynx,myny,
     &shift_up, shift_down, shift_left, shift_right)
      call shift_data(UVEL,mynx,myny,
     & shift_up, shift_down, shift_left, shift_right)
        call shift_data(ZU,mynx,myny,
     & .false., .false.,shift_left,shift_right)
        call shift_data(VP,mynx,myny,
     & shift_up, .false., .false., .false.)
      IL = II1
      if(coords(1) .eq. nprocx-1)  IL  = II2
C
      DO 200  J = 2,JJ1
      DO 200  I = 2,IL
      IF (MASKU(I,J).NE. 0)THEN
      U(I,J) = U(I,J)
     1     + A2(I,J) *ZV(I,J)
C    1     + A24(I,J)*(ZV(I+1,J)+ZV(I+1,J-1)+ZV(I,J+1)+ZV(I,J))
     2     - A31(I,J)*(ZH(I+1,J) - ZH(I,J))
C    2     + A33(I,J)*(HA(I+1,J) - HA(I,J))
     3     + A41     *(UU(I+1,J) + UU(I-1,J) - 2.*UU(I,J))
     4     + A42     *(UU(I,J+1) + UU(I,J-1) - 2.*UU(I,J))
     5     + A5*(TAUX(I,J) - TAUBX(I,J)) - A6*UU(I,J)
```

```
     6     - A71*((ZU(I+1,J)+ZU(I,J))*UVEL(I+1,J)
     7          - (ZU(I,J)+ZU(I-1,J))*UVEL(I-1,J))
     8     - A72* (VP(I,J)          *UVEL(I,J+1)
     9           - VP(I,J-1)        *UVEL(I,J-1))
       ENDIF
  200 CONTINUE
C
```

The data shift routines are then called from the subroutine detailed in the next section.

3.3 The Data Shift Routine

```
   subroutine shift_data(psi,mynx,myny,
     &         shift_up, shift_down, shift_left, shift_right)

        include "mpif.h"
        integer mynx, myny
        real psi(mynx,myny)
        integer i,j,left, right, down, above, comm_2d,strided
        integer ierr, coords(2), stat(MPI_STATUS_SIZE)
        logical shift_up, shift_down, shift_left, shift_right
        common /n1/ comm_2d,coords,left,right,above,down
        common /n2/strided

        if (shift_up) then
        call mpi_send(psi(1,myny-1),mynx,MPI_REAL,above,0,
     &                                comm_2d,ierr)
        call mpi_recv(psi(1,1),mynx,MPI_REAL,down,0,
     &                        comm_2d, stat, ierr)
        endif

        if (shift_down) then
        call mpi_send(psi(1,2),mynx,MPI_REAL,down,1,
     &                               comm_2d, ierr)
        call mpi_recv(psi(1,myny),mynx,MPI_REAL,above,1,
     &                        comm_2d,stat, ierr)
        endif

        if (shift_right) then
        call mpi_send(psi(mynx-1,1), 1, strided, right,2,
     &                                 comm_2d, ierr)
        call mpi_recv(psi(1,1), 1, strided, left, 2,
     &                          comm_2d, stat, ierr)
        endif
```

```
      if (shift_left) then
      call mpi_send(psi(2,1), 1, strided, left, 3,
     &                                    comm_2d, ierr)
      call mpi_recv(psi(mynx,1), 1, strided, right,3,
     &                             comm_2d, stat, ierr)
      endif

      return
      end
```

3.4 Performance Figures on Parallel Platforms

The platforms utilized included the CM5-E (with CMF), the Cray C90 and T90 (with -O3 auto-tasking), the Cray T3E (with HPF and MPI), the SGI Origin2000 (with pfa, HPF and MPI), the IBM SP2 (with HPF) and the Sun Global Works (with HPF). The MPI version of this code employed a 2-D tiling decomposition, and parallel runs were performed up to 512 processors on the T3E, up to 64 processors on the SGI Origin, and up to 64 on the Sun clusters.

indent Figure 3 shows the scalability of the code relative to multiples of 4 CPUs. The T3E with 64 processors achieved an 98 % scaling efficiency relative to 4 CPU's; we found that poor single PE performance held the overall speed down. For 512 processors, the scaling relative to 32 CPUs was 82 %. The Sun systems scaled well, with a 95 % efficiency, up to 32 CPUs, but then their scalability declined sharply. The SGI O2K with 32 processors showed only a 75 % scaling efficiency, but this improved to 84 % with 56 processors. With the O2K, the beneficial cache effects were pronounced with increasing CPUs.

Figure 4 depicts the total MFlops output of the various platforms as a function of CPUS and problem size. For the 2 GB problem size, the O2K efficiency with 64 CPUs is almost 100 % vs. 32 processors, but less than linear for a smaller number. The MFlop output for 56 processors was 3230. For the O2K, the MFlop output for the 4 GB problem size was slightly higher than for the 2 GB size with 16 CPUs; the 4 GB problem has not yet been tested for larger number of processors. We note that the 32 processor MFlop output for the T3E isonly a half of the O2K for 4 GB, but then it scales well up to 64 CPUs.

The auto-tasking versions were quite efficient even for small program sizes (17 Mb) and for small number of processors, with the SGI -pfa compiler option (with -O2 optimization) giving scalings of 1.9, 3.7, and 15.4 for 2, 4 and 16 CPU's, respectively, while the Cray T90 -O3 option (with FT90) gave scalings of 3.6 and 6.6 for 4 and 8 processors, respectively.

4 References

Bleck,R., S.Dean, M.O'Keefe and A.Sawdey, 1995: A Comparison of Data-Parallel and Message-Passing Versions of the Miami Isopycnic Coordinate Ocean Model. Parallel Computing 21, pp.1695-1720.

Blumberg,A.F. and G.L.Mellor, 1987: A Description of a Three-Dimensional Coastal Ocean Circulation Model. In 'Three-Dimensional Coastal Ocean Models', Coastal and Estuarine Sciences, AGU, Washington, D.C., pp.1-16.

Bryan,K., 1969: A numerical method for the study of the circulation of the world ocean. J. Comput. Phys. 4, p.347.

Bryan,K., 1979: Models of the world ocean circulation. Dynamics of Atmospheres and Oceans 3, p.327.

Buzbee,B.L.,F.W.Dorr,L.A.George and G.H.Golub, 1971: The direct solution of the discrete Poisson equation on irregular regions. SIAM J. Numer. Anal. 8, p.722.

Dukowicz,J.K and R.D.Smith, 1994: Implicit free-surface model for the Bryan- Cox-Semtner ocean model. J. Geophys. Res. 99, p.7991.

Dukowicz,J.K, R.D.Smith and R.C.Malone, 1993: A reformulation and implementation of the Bryan-Cox-Semtner ocean model on the connection machine. J. Atmos. Ocean. Technol. 10, p.195.

Kantha,L.H., 1995: Barotropic Tides in the Global Ocean from a Nonlinear Tidal Model Assimilating Altimetric Tides, Part I. Model Description and Results. J. Geophys. Res. 100, pp.25283-25308.

Ma,H., J.McCaffrey and S.Piacsek, 1998: A Parallel Implementation of a Spectral Element Ocean Model for Simulating Low-Latitude Circulation System. in Parallel Computational Fluid Dynamics, pp.641-648, ed. D.R.Emerson, A.Ecer, J.Periaux, N.Satofuka and P.Fox. Elsevier Press.

Madala,R.V. and S.A.Piacsek, 1977: A Model for Baroclinic Oceans. J. Comp. Phys. 23, p.167.

Piacsek,S.A. and R.Allard, 1994: Barotropic Coastal Currents in the Mediterranean. In Proceedings of the Second International Conference on Air-Sea Interaction and on Meteorology and Oceanography of the Coastal Zone. American Meteorological Society, Boston, MA, p.206.

Oberhuber,J.M. and K.Ketelsen, 1995: Parallelization of an OGCM on the CRAY T3D. Internal report, Deutsches Klimarechenzentrum GmbH, Model Support Group.

Piacsek,S.A. and A.J.Wallcraft, 1993: Initial Experiences with the Connection Machine at NRL. NRL Technical Note # 73-5089-03.

Schwarztrauber,P.N., 1977: The methods of cyclic reduction, Fourier analysis and cyclic reduction, and Fourier analysis - for the discrete solution of Poisson's equation on a rectangle. SIAM Rev. 19, p.490.

Smith,R.D., J.K.Dukowicz and R.C.Malone, 1992: Parallel ocean general circulation modeling. Physica D Amsterdam 60, p.38.

Wallcraft,A.J., 1991: The Navy Layered Ocean Model Users Guide. NOARL Report # 35, Dec. 1991, 21 pp. Stennis Space Center, MS 39529, USA

Wallcraft,A.J. and D.R.Moore, 1997: The NRL layered ocean model. Parallel Computing 23, pp. 2227-2242.

Webb,D.J., 1995: An ocean model code for array processor computers. Internal Document No.324, Institute of Oceanographic Sciences, Wormley, U.K.

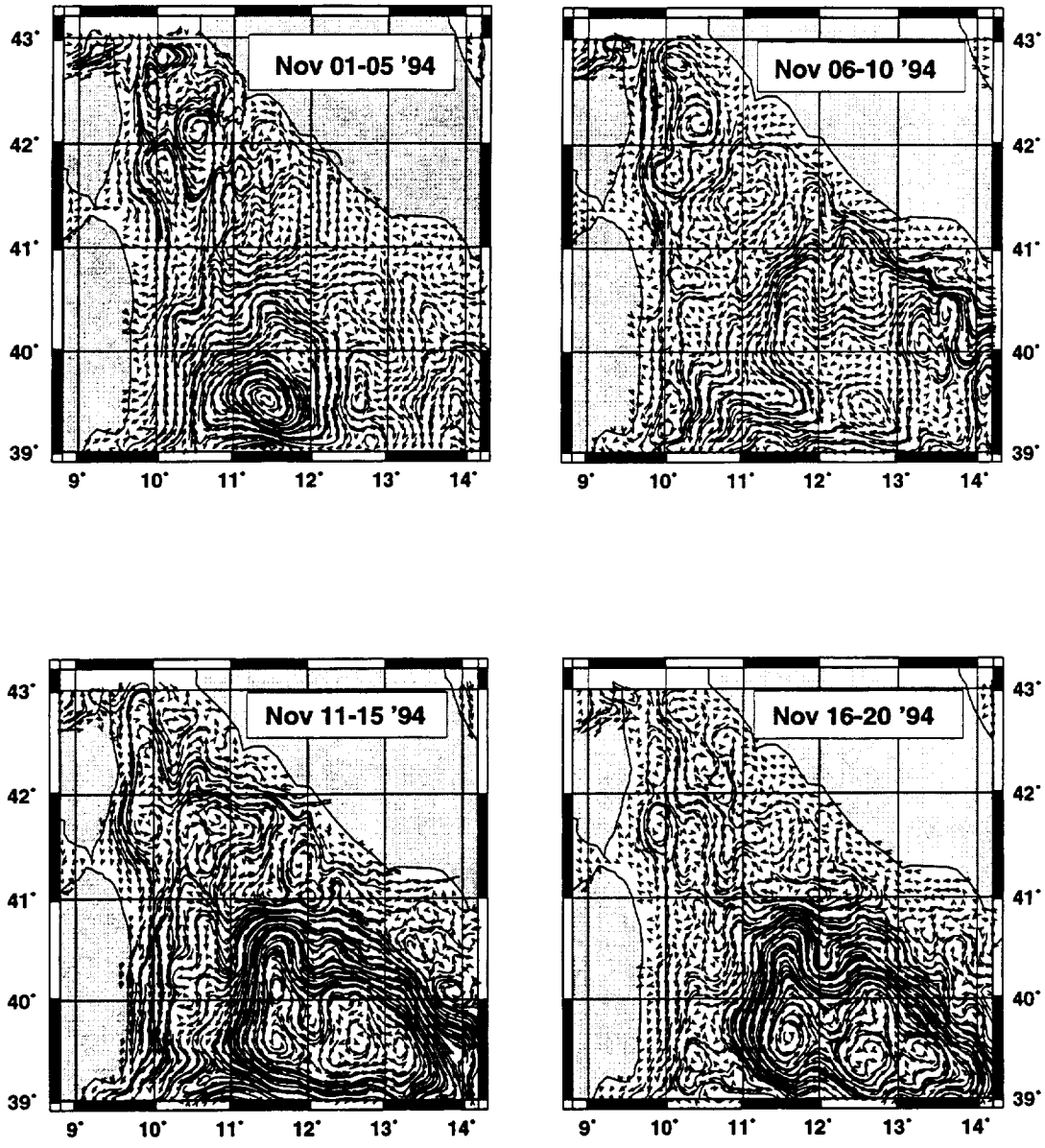


Figure 1: The barotropic circulation in the Tyrrhenian Sea, for the month November 1994.

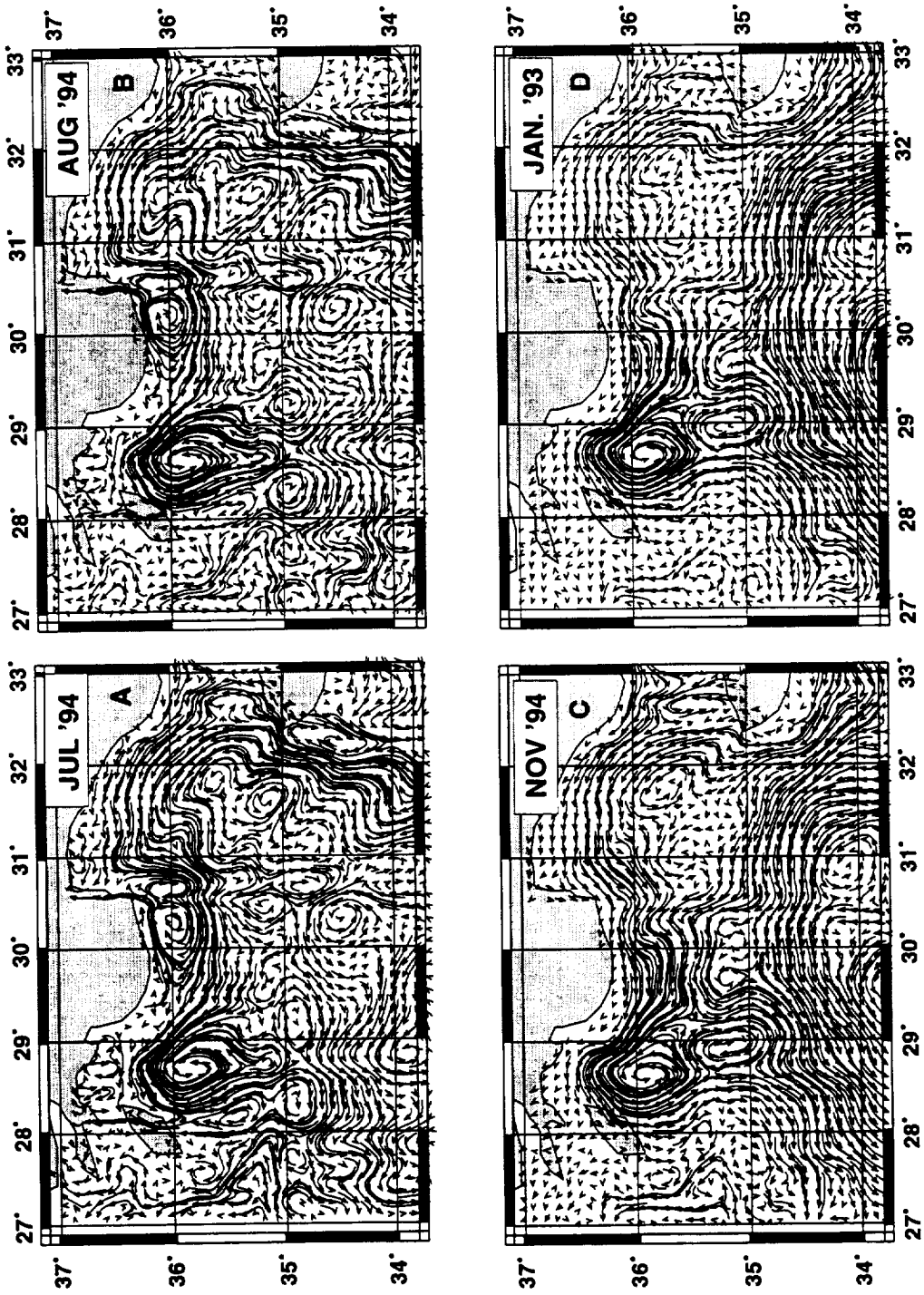


Figure 2: Seasonal evolution of the barotropic circulation in the NE quadrant of the Eastern Mediterranean, for 1994.

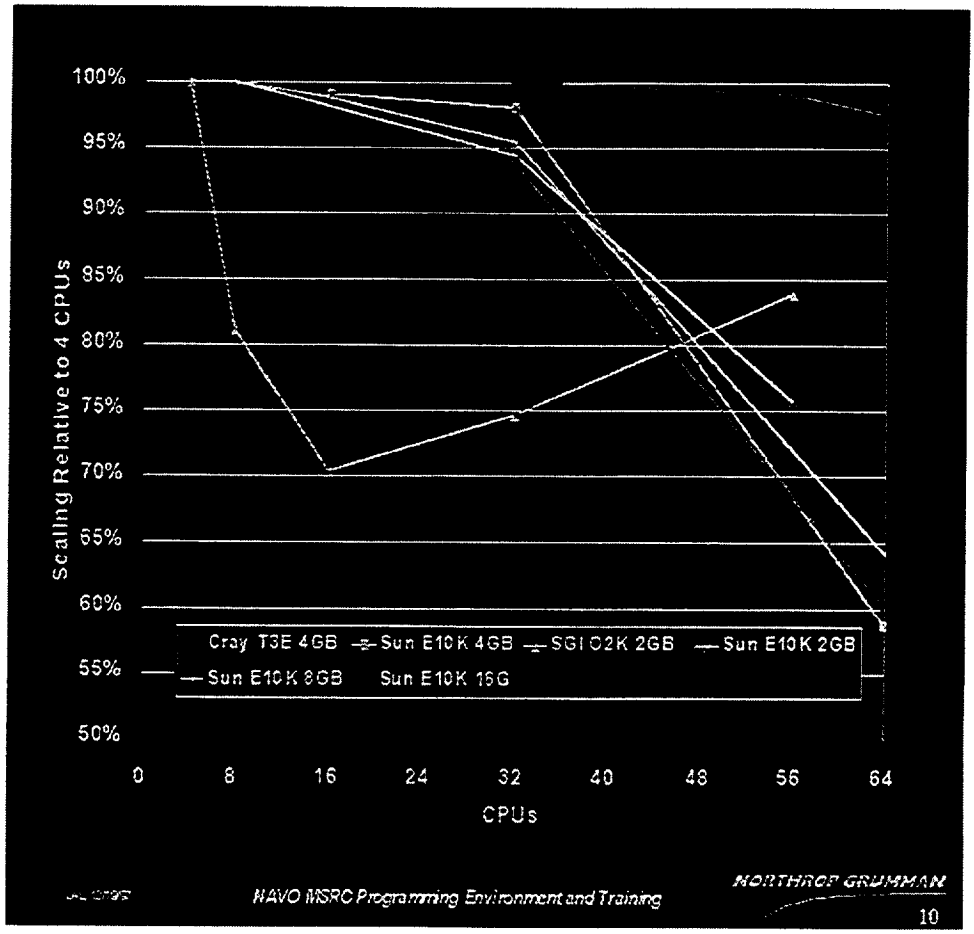


Figure 3: Scalability of the MPI version of the barotropic code relative to 4 CPUs

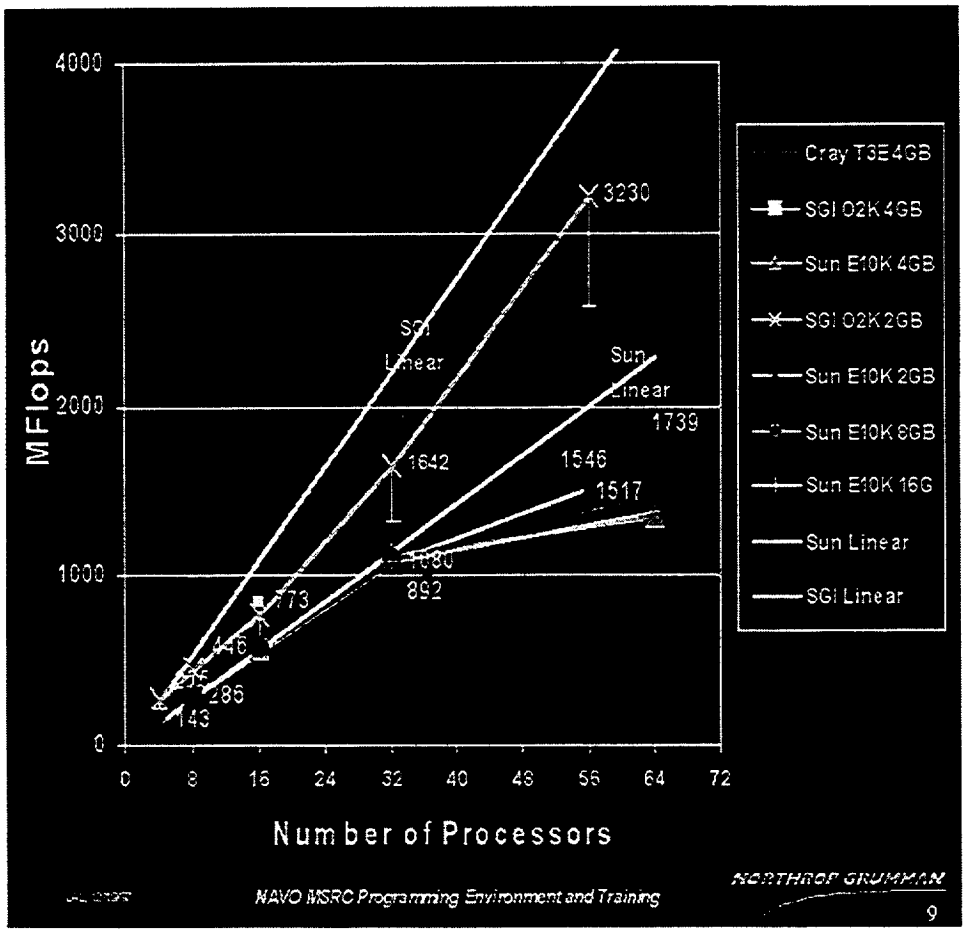


Figure 4: Scalability of the MPI version of the barotropic code in terms of MFlops

Author: Mohamed Iskandarani
Institute of Marine and Coastal Sciences
Rutgers University
71 Dudley Road
New Brunswick, NJ 08901-8521
mohamed@ahab.rutgers.edu

Parallel Performance of a 3D Spectral Element Ocean Model

In the present work, we investigate the scalability and parallel performance of a new three-dimensional spectral element ocean model. This new model solves the hydrostatic and Boussinesq primitive equations. It features a spectral element discretization in all three space dimensions with an unstructured grid in the horizontal, and a terrain-following structured discretization in the vertical direction. The computational tasks in the new model consist of time-integrating the barotropic component of the flow, calculating the three-dimensional tendencies, solving the implicit system of equations, and updating the diagnostic variables of pressure, density and vertical velocity. Here, we present our parallel implementation of the above tasks. We also present our analysis of several numerical experiment in order to identify the break-down of the computational cost among the tasks listed above. The numerical experiments also serve to illustrate the scalability and performance of the model in a typical basin-scale oceanic simulation.

MASSIVELY PARALLEL IMPLEMENTATION OF A HIGH ORDER DOMAIN DECOMPOSITION EQUATORIAL OCEAN MODEL

Hong Ma
Brookhaven National Laboratory, Upton, NY 11973
hm@bnl.gov 516 344-4138

Joseph W. McCaffrey **Steve Piacsek**
Naval Research Laboratory, Stennis Space Center, MS 39529
mccaffrey@nrlssc.navy.mil piacsek@nrlssc.navy.mil
601 688-5053 601 688-5316

Abstract

The present work is about the algorithms and parallel constructs of a spectral element equatorial ocean model. It shows that high order domain decomposition ocean models can be efficiently implemented on massively parallel architectures, such as the Connection Machine Model CM5. The optimized computational efficiency of the parallel spectral element ocean model comes not only from the exponential convergence of the numerical solution, but also from the work-intensive, medium-grained, geometry-based data parallelism. The data parallelism is created to efficiently implement the spectral element ocean model on the distributed-memory massively parallel computer, which minimizes communication among processing nodes. Computational complexity analysis is given for the parallel algorithm of the spectral element ocean model, and the model's parallel performance on the CM5 is evaluated. Lastly, results from a simulation of wind-driven circulation in low-latitude Atlantic ocean are described.

1 Introduction

The spectral element method is a combination of both the spectral and the finite element methods. The spectral element method is also called the "p-type finite element" method, or the h-p type weighted residual method. Like the spectral method, it uses high order polynomials as trial functions, but like the finite element method, it decomposes the computational domain into many elements and defines local trial functions. The hybrid character of the spectral element method enables it to overcome the shortcomings of both the spectral method and the finite element method but still retain their advantages. Since the trial functions of the spectral element method are local, it can handle complex geometry easily. On the other hand, it is still a high order weighted residual method, so the exponential convergence rate is achieved as the degree of the polynomials in each element is increased. The main difference between the spectral element method and the spectral multi-domain method is that the C^0 and C^1 boundary conditions at the interface of the elements have to be explicitly enforced by the spectral multi-domain method. The spectral element method, by contrast, uses the variational principle to guarantee C^0 and C^1 (weakly) continuity at the

interface, which results in a much simpler and more natural approach than the nonvariational method; therefore, parallel algorithms can be conveniently implemented. In the past decade or so, research on the spectral element method has made important progress in perfecting this state-of-the-art numerical method [4,13,14]. More recently, the spectral element method has shown encouraging potential in oceanic applications [5,6,7,8,9,10,11].

The present work is about the implementation and results of a massively parallel, spectral element, high-resolution, three-dimensional equatorial ocean model which, in particular, is capable of resolving both the horizontal and the vertical structures of the low-latitude western boundary processes. The current version of the model is driven solely by wind stress and ignores the dynamical effects of stratification. This model is designed to study the effect of wind in the formation and variation of important meso-to-small scale equatorial ocean phenomenon, such as eddies, low-latitude western boundary currents, and vertically alternating equatorial zonal jets. The high efficiency of this model is based on an optimized coupling between the numerical algorithm and the computer architecture (algorithm-architecture). In addition to its exponential convergence rate, the model's performance is further enhanced by the spectral element tensor-product factorization and spectral element parallelism.

2 Governing Equations

The model equations include three dimensional time dependent primitive equations with hydrostatic approximation [1]. The vertical Coriolis force term in the zonal momentum equation, which is usually omitted in general circulation models, is kept for the reason that it could become important at the equator.

$$\frac{du}{dt} - (2\Omega + \frac{u}{r \cdot cos\phi})(v \cdot sin\phi - w \cdot cos\phi) =$$
$$-\frac{1}{\rho \cdot r \cdot cos\phi}\frac{\partial p}{\partial \lambda} + A_H \Delta u + A_v \frac{1}{r^2}\frac{\partial}{\partial r}(r^2 \frac{\partial}{\partial r})u \tag{1}$$
$$\frac{dv}{dt} + \frac{wv}{r} + (2\Omega + \frac{u}{r \cdot cos\phi})u \cdot sin\phi =$$
$$-\frac{1}{\rho \cdot r}\frac{\partial p}{\partial \phi} + A_H \Delta v + A_v \frac{1}{r^2}\frac{\partial}{\partial r}(r^2 \frac{\partial}{\partial r})v \tag{2}$$
$$-\frac{1}{\rho}\frac{\partial p}{\partial r} - g = 0 \tag{3}$$
$$\frac{1}{r \cdot cos\phi}\frac{\partial u}{\partial \lambda} + \frac{1}{r \cdot cos\phi}\frac{\partial (v \cdot cos\phi)}{\partial \phi} + \frac{\partial w}{\partial r} = 0 \tag{4}$$

where

$$\frac{d}{dt} = \frac{\partial}{\partial t} + \frac{u}{r \cdot cos\phi}\frac{\partial}{\partial \lambda} + \frac{v}{r}\frac{\partial}{\partial \phi} + w\frac{\partial}{\partial r} \tag{5}$$

$$\Delta = \frac{1}{r^2 cos^2\phi}\frac{\partial^2}{\partial\lambda^2} + \frac{1}{r^2 cos\phi}\frac{\partial}{\partial\phi}(cos\phi\frac{\partial}{\partial\phi}) \tag{6}$$

and λ is longitude, ϕ latitude, and r radial distance; u, v, and w are the velocity components in the λ, ϕ, and r directions, respectively; p is the pressure term. A_H and A_v are the eddy viscosity coefficients in the horizontal and vertical directions, respectively.

For convenience, the original spherical coordinates (λ, ϕ, r) are mapped onto another coordinates system (x, y, z), and relationship between these two coordinate systems is defined as:

$$x = r_0 cos\phi \tag{7}$$
$$y = r_0\phi \tag{8}$$
$$z = r - r_0 \tag{9}$$

where r_0 is the radial distance of the sea level.

No-normal flow, no-slide boundary conditions are applied to all lateral boundaries and to the ocean floor. At the sea-surface, we assume the rigid-lid boundary condition. The present model is solely driven by wind stress:

$$\rho_0 A_v \frac{\partial u}{\partial z}|_{z=0} = \tau_x \tag{10}$$
$$\rho_0 A_v \frac{\partial v}{\partial z}|_{z=0} = \tau_y \tag{11}$$

where τ_x and τ_y are surface wind stress components.

3 Spectral Element Discretization and Solvers

The basis sets used in the present work are as follows:

$$\psi^e_{lmn}(\xi, \eta, \zeta) = h_l(\xi)h_m(\eta)h_n(\zeta) \qquad l, m, n \in \{0, 1, ...N\}^3 \tag{12}$$

where $h_i(s)$ are the Gauss-Lobatto-Legendre polynomials.

If we use a single subscript, q ($q \in \{1, 2, ..., (N+1)^3\}$), the mapping between a macroelement, Ω_e, and its phase domain, $\hat{\Omega}_e$, can be expressed as:

$$\mathbf{x} = \mathbf{x_q}\psi^e_q(\xi) \tag{13}$$

Where $\mathbf{x} \in \Omega^e$ and $\xi \in \hat{\Omega}_e$.

Let solution $\mathbf{u}$ at time $n\Delta t$ on each subdomain Ω^e be expanded as:

$$\mathbf{u}^e(\mathbf{x}, \mathrm{n\Delta t}) = \mathbf{u}^e_q(\mathrm{n\Delta t})\psi^e_q[\xi(\mathbf{x})] \tag{14}$$

where $f_q^e(t)$ is the value of function f at the collocation point $\mathbf{x}_q \in \Omega^e$ at time t.

By using the same variational procedures as those in [6,7], i.e., all the integrations are evaluated by the Gauss-Lobatto quadrature scheme, which is an exact formula for $(2N-1)^{th}$ order polynomials, the spatially discretized formulae for the primitive equations can be obtained. In particular, the isoparametric spectral element discretization formulae for the horizontal momentum equations of the present primitive equation model are virtually identical as those in [6, 7]. One advantage of using the Gauss-Lobatto-Legendre polynomials as basis functions is that we only have to deal with one set of grid points for both interpolating the solutions and evaluating the integrals.

We choose the isoparametric spectral element discretization scheme [6, 7], namely, using nonstaggered grids, for the present numerical model. The nonstaggered formulation avoids spurious pressure modes as staggered schemes do, and, at the same time, has the advantage that pressure is continuous across boundaries of the spectral elements. Only one set grids is required for both interpolation and quadrature, hence simplifying operations.

The discretized incompressibility condition and the hydrostatic condition have the following format:

$$[D^z][w] = [g]$$

where $[D^z]$ is the matrix generated by applying variational procedures to the vertical differentiation operator; [w] is the vector representing the unknown at the collocation points, and [g] is a vector whose components are known.

To obtain the solution for the vertical velocity, w, we need to solve a matrix problem of the above format. It can be done by using either matrix iteration methods or direct matrix inversion. The latter is especially efficient when the vertical grains of the spectral element mesh are parallel to the z axis, since the dimension of the matrix to be inverted is the same as the number of levels in the vertical direction.

The time marching scheme for the hyperbolic equations of the present model is the 3^{rd} order Adams-Bashforth scheme. This scheme has proven to be efficient in high Reynolds number, high resolution simulations, especially in a massively parallel computing environment [7]. In fact, except in the upper range of eddy viscosity (diffusion) for oceanic applications, it is likely to be more efficient to use a fully-explicit scheme because it results in diagonal stiffness matrices for the hyperbolic equations.

A preconditioned conjugate gradient iterative solver is used in the present model to solve the elliptic equation $[A][x] = [b]$ associated with the pressure term, which has the following algorithm

$$[x_0] = initial \quad guess; \quad [r_0] = [b] - [A][x_0]; \quad [q_0] = [P^{-1}][r_0]; \quad [s_0] = [q_0];$$
$$\alpha_m = [r_m] \cdot [s_m]/[q_m] \cdot \{[A][q_m]\}; \quad [x_{m+1}] = [x_m] + \alpha_m[q_m];$$
$$[r_{m+1}] = [r_m] - \alpha_m[A][q_m]; \quad [s_{m+1}] = [P^{-1}][r_{m+1}];$$

$$\beta_m = [r_{m+1}] \cdot [s_{m+1}]/[r_m] \cdot [s_m]; \quad [q_{m+1}] = [s_{m+1}] + \beta_m [q_m] \tag{15}$$

where P is the Jacobi preconditioner.

Evaluation of the matrix-vector product, $[A][u]$, constitutes the operation count kernel of the spectral element iteration solver. Inherited from the properties of the spectral method, the number of operations for the matrix-vector product $[A][u]$ in the present spectral element model would be proportional to KN^{2d} (d is the number of spatial dimensions) if a simple-minded algorithm were applied. Although the problem is less severe for the spectral element method than it is for the spectral method, it still hinders the efficiency of the 3-D model. Fortunately, the partial summation algorithm, which was first proposed by Orszag (1980) for the spectral method, can drastically reduce the cost of solving the matrix-vector product problem.

Assume $[A]$ is the global stiffness matrix for the laplacian operator. The straightforward spectral element formulation gives the tensor product form

$$[A][u](q,r,s) = \sum_{k=1}^{K}{}' \sum_{l=0}^{N} \sum_{m=0}^{N} \sum_{n=0}^{N} A^k(q,r,s,l,m,n) u^k(l,m,n)$$
$$(q,r,s \in \{0,...N\}^3) \tag{16}$$

where $[A][u](q,r,s)$ are elements of $[A][u]$; $A^k(q,r,s,l,m,n)$ are elements of the local stiffness matrix $[A^k]$ and $\sum'$ is the direct stiffness matrix summation. The evaluation of $[A][u]$, therefore, requires $O(KN^6)$ operations.

The present work uses the partial summation algorithm at the elemental level to evaluate $[A][u]$ more efficiently. $A^k(q,r,s,l,m,n)$ can be evaluated in the form of

$$A^k(q,r,s,l,m,n) = H^x(l,q) H^y(m,r) H^z(n,s) \tag{17}$$

where $H^p(i,j)$ are functions of i and j.

Therefore, the auxiliary matrices $[B]$ and $[C]$ can be constructed in the following way:

$$B(l,m,s) = \sum_{n=0}^{N} u^k(l,m,n) H^z(n,s) \tag{18}$$

$$C(l,r,s) = \sum_{m=0}^{N} B(l,m,s) H^y(m,r) \tag{19}$$

Then $[A^k][u^k]$ can be evaluated by

$$[A^k][u^k](q,r,s) = \sum_{l=0}^{N} C(l,r,s) H^x(l,q) \qquad q,r,s \in \{0,1,...,N\}^3 \tag{20}$$

Now, the total operation count in computing B, C, and $[A][u]$ is $O(KN^4)$; this is N^2 times less expensive than using the straightforward algorithm to evaluate the matrix-vector product.

The greatly reduced operation count resulting from the partial summation algorithm is not the only advantage of the spectral element model in evaluating the matrix-vector product; it also has a much smaller storage requirement. In fact, the storage required to evaluate $[A][u]$ for a 3-D model would be $O(KN^3)$ instead of $O(K^2N^6)$ because the stiffness matrix elements are "computed on the fly" rather than stored in the memory.

4 Parallel Implementation

The spectral element primitive equation ocean model is parallelized to run efficiently on the Connection Machine Model CM5. In order to avoid unnecessary communication among processing nodes, which is of first order importance in a parallel implementation on a distributed memory, massively parallel architecture, a data mapping scheme was created so that all the information related to a given spectral element is collected in the memory of a single processor. Prior to assembling the global stiffness-matrices, only data related to a given spectral element are used to create the local matrices of that spectral element. At this stage, all computations are carried out at the local level, therefore, there is no communication among neighboring processors while assembling local (elemental) matrices. On the Connection Machine model CM-5, we pursue data parallelism by designing the layout of the arrays of the spectral element model in such a way that the axes along the number of elements are assigned as parallel dimensions, and those along intra-element degrees of freedom as serial ones. is no communication among neighboring processors during elemental level computations, and they are performed concurrently across all virtual processors.

The computational kernel of solving the discretized primitive equations model is calculating matrix-vector products. We split the procedure of calculating matrix-vector products into two steps, each of which admits concurrency. At the first step, the matrix-vector products are carried out at the elemental level with, for example, the elemental Laplacian and mass matrices:

$$r^k(i) = \sum_{q=1}^{(N+1)^d} A^k(i,q)u^k(q) \qquad i \in \{1,2,...,(N+1)^d\}, \quad k \in \{1,2,...,K\} \tag{21}$$

$$s^k(i) = B^k(i)u^k(i) \qquad i \in \{1,2,...,(N+1)^d\}, \quad k \in \{1,2,...,K\} \tag{22}$$

After applying tensor-product factorization, the computational complexities to evaluate (21) and (22) would be C_1KN^{d+1}/Q and C_2KN^d/Q, respectively, where Q is the number of physical processors involved. On the Connection Machine systems, parallel data structure allows (22) to be performed in an array operation, which means that

thousands of simultaneous multiplications are made across all the array elements. Hence, C_2 is a small number. Consequently, diagonal preconditioning is especially efficient in the data parallel environment: it does not require direct stiffness summation, and only local computation is involved. Iteration counts can be reduced by twenty to thirty percent with about a one percent increase in cost. The processing nodes on the CM-5 model are equipped with powerful vector-processing units that can further reduce the cost of elemental level computation. These vector-processing units are most efficient when the order of the spectral elements is high.

The second step is to carry out direct stiffness summation, $\sum_{k=1}^{K}{}'$, in which contributions from local nodes that share the same physical coordinates are first accumulated, and then assigned back to those local nodes. In a serial spectral element model, this procedure can be accomplished by using global and local index systems, and is automatically done as the matrix computation is made for each spectral element. In the parallel spectral element model, however, it is more efficient to use a separate step for the direct stiffness summation. Since each spectral element has at least one edge (two-dimensional case) or one surface (three-dimensional case) that is shared by a neighboring element, the direct stiffness summation can be carried out simultaneously along these edges or surfaces enabling structured message exchange, i.e., edge-based message exchange for two-dimensional problems, and surface-based message exchange for three-dimensional ones. Since this kind of information exchange takes place along the linkages of the "macro-element-skeleton", it can be easily synchronized for all elements in the entire domain. The work per processor that is required in this procedure is $C_3 dKN^{d-1}/Q$. The structured message exchange mostly avoids explicit short messages, and it considerably improves the parallel efficiency of the spectral element model.

With parallel prefix of the CM Fortran, MATMUL and SUM, an inner product can be executed completely in parallel. Its computational complexity is $C_4 KN^d/Q$. Due to the high level of concurrency afforded by the parallel prefixes, C_4 is a small number.

In high Reynolds number case, the discretized horizontal momentum equations only requires a direct method to solve. The computation kernel here is the evaluation of the advection term where concurrency can be achieved at different levels of the computation. We first evaluate the shears of velocities at all nodal points

$$\frac{\partial u_m^{n,e}(l)}{\partial x_j} = \sum_{s=1}^{(N+1)^d} u_m^{n,e}(s) \frac{\partial \psi_s^e}{\partial x_j}$$
$$l \in \{1, 2, ..., (N+1)^d\}, \quad m, j \in \{1, 2, ..., d\}^2, \quad e \in \{1, 2, ..., K\} \quad (23)$$

This operation is executed concurrently across all virtual processors. With the partial summation method, the computational complexity for (23) is $C_5 K(N+1)^{d+1}/Q$. The discretized advection term also can be written as

$$C^{n,e}_{m,j}(p,q) = \sum_{s=1}^{(N+1)^d} u^{n,e}_m(s) \frac{\partial \psi^e_s(q)}{\partial x_j} \int_{\hat{\Omega}^e} \psi^e_p \psi^e_q |J^e| d\hat{\Omega}^e$$
$$p,q \in \{1,2,...,(N+1)^d\}^2, \quad j,m \in \{1,...,d\}^2 \qquad (24)$$

Therefore, once the shears of velocities are obtained, the remaining operation to evaluate the advection terms is the same as that of (22) Hence, the total computational complexity of (24) is $C_2K(N+1)^d/Q + C_5K(N+1)^{d+1}/Q$.

As spectral elements are of high-order, most of the costly operations are at the elemental level, and they are executed concurrently. The spectral granularity at the elemental level can take full advantage of the computing power that the latest processing units provide. The structured message exchange, combined with parallel prefix, makes inter-element communication a lower-order rather than a highe-order cost, compared to that of elemental level computation. This communication cost should be much smaller than that of the h-type finite element model, partially because many fewer redundant nodel values, shared by more than one element, have to be stored.

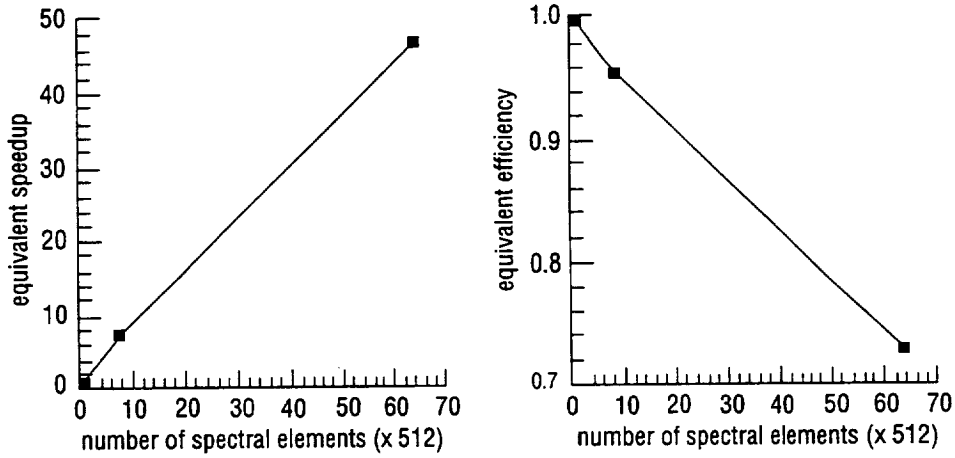


Figure 1. Parallel performance measure on a CM5 partition with 256 processors. $N = 6$.

In the parallel implementation of the present spectral element model on the CM5, the number of virtual processors always equals to the number of spectral elements. Therefore, we can use "equivalent speedup"$= (K * T_1)/T_K$ and "equivalent efficiency"$= T_1/T_K$ to measure the parallel performance of the spectral element primitive equation model, where T_K is the CM5 cpu time per time step with K spectral elements. Since on a serial computer, the computational cost of the spectral element primitive ocean model is proportional to the number of spectral elements, $K * T_1$ is roughly how much time it would take to execute one time step if the CM5 had only one processor. Figure 1 shows that with a fixed number of physical processors, the performance of the spectral element primitive equation model scales very well until the

number of spectral elements becomes so large that the memory in the CM5 partition is saturated. The excellent scalability of the model recovers when the size of the CM5 partition is increased.

5 Results From An Equatorial Atlantic Experiment

In this numerical experiment, the model is driven solely by surface wind stress, and the model ocean is an idealized rectangular basin. It is designed to study the effect of wind in the formation and variation of important equatorial ocean phenomena which are meso-scale at least in one spatial dimension, such as low-latitude western boundary currents, eddies, and vertically/meridionally alternating low-latitude zonal jets. The assigned values for the horizontal and vertical eddy viscosity coefficients for the current simulation are $2.4 * 10^6\ cm^2 s^{-1}$ and $30\ cm^2 s^{-1}$, respectively. Since the present spectral element model has adequate resolution to resolve meso-scale eddies, we were able to use a horizontal eddy viscosity coefficient which is an order of magnitude smaller than what was typically used in OGCMs to allow the meso-scale processes to be modeled more realistically.

The lengths of the edges of the macro spectral elements which were used to discretize the present model measured 3.5° in the meridional direction, and 90 m vertically. A strip of refined spectral elements is embedded in the region west of 34° W where the zonal length of the macro spectral elements measured 1.5°; elsewhere it measured 6°. This discretization strategy is based on the fact that short equatorial waves are confined near to the western boundary region. The seventh order Gauss-Lobatto-Legendre polynomials were used to construct the basis functions within each macro spectral elements. Based on the role-of-thumb that 3.5 interpolation points per wavelength would be required for the spectral element model to resolve a wave-like solution with O(1%) numerical error, the present spectral element ocean model configuration can adequately resolve waves of wave-lengths 150 km in the meridional direction, 50 m in the vertical direction, and 70 km in the zonal direction (300 km if east of 34° W).

Figures 2a and 3a show that beneath the surface layer, there are bands of westward currents centered around 10°N and 9°S, which supply the equatorward western boundary undercurrents. Compared with the observed structure of the intermediate layer currents [2], these model subsurface westward currents correspond to the North and the South Equatorial Currents (NEC and SEC), respectively. In the winter season, the model North Equatorial Undercurrent (NEUC) at 3°N is fed from the north by the southward western boundary undercurrent, supplied by the NEC. In summer season, the southward western boundary undercurrent does not reach as far south as in winter, and it veers into the North Equatorial Countercurrent (NECC) centered at 7°N. In the upper part of the undercurrent layer ($\sim$ 100 m deep), in all seasons, the SEC feeds the model North Brazil Current (NBC) which in turn feeds the South Equatorial Undercurrent (SEUC) near 3°S and the EUC. The model NBC also feeds the NEUC in summer, in contrast to the northerly supply of the NEUC in winter. An interesting phenomenon is that while the model NEUC is a permanent current

feature, its supply in the western boundary region alternates between a southerly and northerly one depending on the season. From Figures 2a and 3a, we note that outside the western boundary region, the EUC provides input to the NEUC and the SEUC through poleward meridional flows on both of its sides. The present model reproduces the retroflection of the NBC near 4°N in summer. In the upper undercurrent layer, the NBC curves back to the NEUC (Figure 3a).

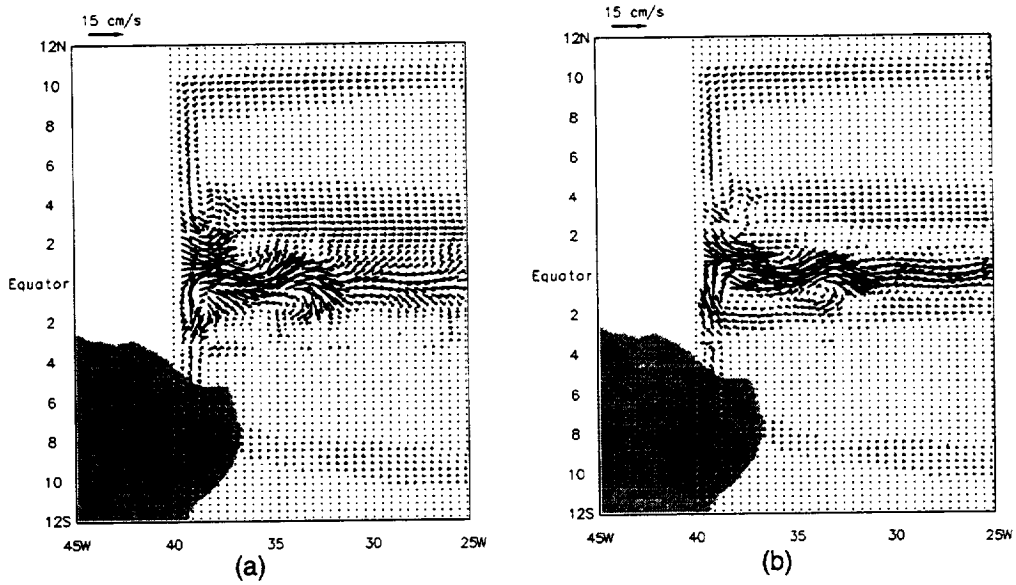


Figure 2. Model currents in January. (a) 100 m. (b) 200 m.

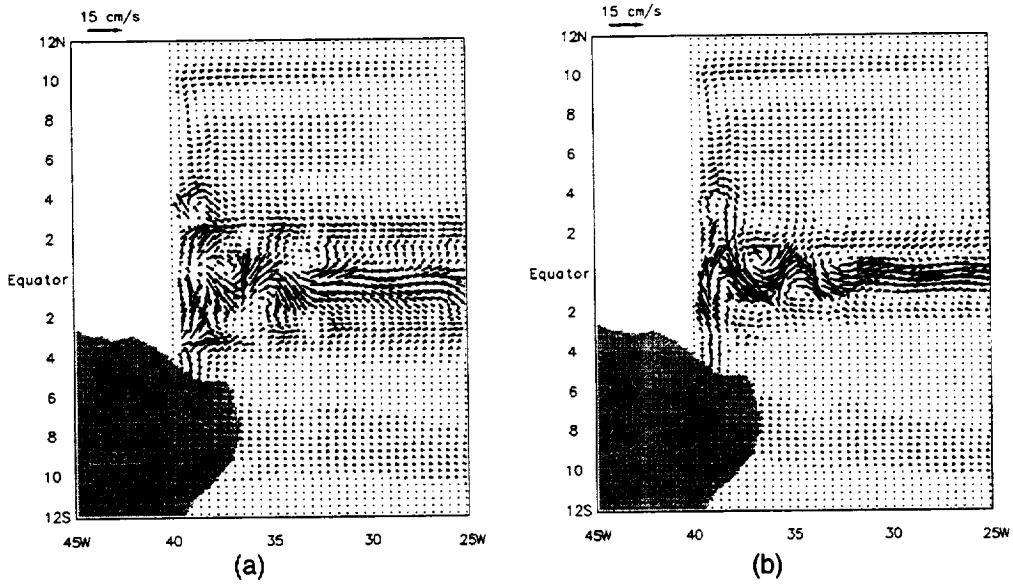


Figure 3. Model currents in July. (a) 100 m. (b) 200 m.

In the lower undercurrent layer ($\sim$ 200 m), the eastward model NEUC and SEUC diminish, and the model Equatorial Undercurrent (EUC) is now flanked by two westward off-equatorial undercurrents located at 1.5°N and at 1.5°S (Figures 2b and 3b). In winter, the westward off-equatorial current north of the equator diminishes east of 35°W (Figure 2b). Also, in summer at this vertical level, the equatorward (southward) off-shore countercurrent, which is part of the NBC retroflection, extends all the way to the equator where it joins the EUC. At 100 m depth, however, this countercurrent merges instead with the NEUC (Figure 3a). The model NBC does not retroflect in winter. In the lower part of the upper-layer (below 200 m), however, there is an offshore anticyclonic gyre at the same latitude where the NBC retroflects in summer (Figure 2b). The temporal and spatial features of the model upper ocean currents described in this section closely resemble those of their counterparts observed in the tropical Atlantic, which are summarized in [2].

It is interesting that the branching of the model NBC is strongly layered. In the near-surface layer, the NBC feeds the NECC between 6°N and 9°N in winter, and between 4°N and 7°N in summer. Below the near-surface layer, the model NBC originates south of the equator. In the layer between 60 m and 80 m depth, the NBC branches off mostly to the SEUC near 4°S ; in the layer between 150 m and 250 m, it branches mostly to the EUC; and in the intermediate layer between these two, it supplies both the SEUC and the EUC. In summer season, at depths below 400 m, the model NBC does not branch off to eastward interior flows until it reaches near 4°N where it retroflects into an equatorward countercurrent which eventually joins the model EUC. This layered separation pattern of the NBC was also reported in water mass studies, e.g.,[3,12].

The fact that the present model reproduced all major features of the currents in the upper couple hundred meters of the tropical Atlantic Ocean suggests that the wind effect here is the deterministic mechanism of current formation an d variations. More details of the present numerical simulation are described in [11].

6 Conclusions

The present work shows that high order domain decomposition ocean models can be efficiently implemented on massively parallel architectures, such as the Connection Machine Model CM5. The optimized computational efficiency of the parallel spectral element ocean model comes not only from the exponential convergence of the numerical solution, but also from the work-intensive, medium-grained, geometry-based data parallelism. The data parallelism is created to efficiently implement the spectral element ocean model on the distributed-memory massively parallel computer, which minimizes communication among processing nodes and results in a highly scalable performance. The same advantage of the nonstaggered grid formulation was found in the present parallel, three dimensional, spectral element ocean model as in the earlier spectral element shallow water equation model [6,7].

Acknowledgement

This work was supported in part by DoD Common High Performance Computing Software Support Initiative (CHSSI) and by the U. S. Department of Energy under Contract No. DE-AC02-98CH10886.

References

1. Bryan, K., Mon. Wea. Rev., 97 (1969) 806.
2. Bub, F. L. and W. S. Brown, J. Geophys. Res., 101 (1996) 11903.
3. Cochrane, J. D., F. J. Kelley, and C. R. Olling, J. Phys. Oceanogr., 9 (1979) 724.
4. Fischer, P. F. and A. T. Patera, J. Comput. Phys., 92 (1991) 380.
5. Ma, H., J. of Marine Research, 50 (1992) 567.
6._____, J. Comput. Phys., 109 (1993) 133.
7._____, Brookhaven National Laboratory Technical Report (1994) BNL-61103.
8._____, in Parallel Computational Fluid Dynamics: Implementations and Results Using Parallel Computers (A. Ecer, J. Periaux, N. Satofuka and S. Taylor, eds.) 239. Elsevier, North-Holland, 1996.
9._____, J. Mar. Res., 54 (1996) 35.
10. Ma, H., J. McCaffrey, and S. Piascek, Proceedings of Parallel CFD '97 (1997)641. Manchester, UK.
11. _____ (1998), submitted to J. Mar. Res..
12. Metcalf, W., J. Mar. Res., 26 (1968) 232.
13. Patera, A. T., J. Comput. Phys., 54 (1984) 468.
14. Ronquist, E. M., Optimal Spectral Element Methods for the Unsteady Three Dimensional Navier-Stokes Equations, Ph.D. Thesis, The Massachusetts Institute of Technology (1988).

The Los Alamos Coupled Climate Model

Philip W. Jones
Theoretical Division, Los Alamos National Laboratory
T-3 MS B216, Los Alamos, NM 87545
pwjones@lanl.gov, Phone: +1 505 667-6387, FAX: +1 505 665-5926
Robert C. Malone
Advanced Computing Laboratory, Los Alamos National Laboratory
MS B287, Los Alamos, NM 87545
rcm@lanl.gov, Phone: +1 505 667-5925, FAX: +1 505 665-4939
C. Aaron Lai
Earth and Environmental Sciences Division, Los Alamos National Laboratory
MS B401, Los Alamos, NM 87545
cal@lanl.gov, Phone: +1 505 665-6635, FAX: +1 505 665-3415

Abstract

A climate model which couples ocean, sea ice, atmosphere and land components is described. The component models are run as autonomous processes coupled to a flux coupler through a flexible communications library. Performance considerations of the model are examined, particularly for running the model on distributed-shared-memory machine architectures.

I. Introduction

To gain a full understanding of the Earth's climate system, it is necessary to understand physical processes in the ocean, atmosphere, land and sea ice. In addition, interactions between components are very important and models which couple all of the components into a single coupled climate model are required. A variety of such models have been developed using quite different approaches. For example, the Geophysical Fluid Dynamics Laboratory (GFDL) coupled model [1] is a single integrated model which is run at very coarse resolution for many thousands of years. At the other end of the spectrum is the Climate System Model (CSM) [2] at the National Center for Atmospheric Research (NCAR) which couples different models running autonomously at moderate resolution. Our ultimate goal is to produce a coupled model which can be used for century-scale climate simulations at resolutions that will allow us to resolve eddy processes which are important in ocean dynamics.

For high-resolution climate simulations, it is important to develop a coupled climate model that runs efficiently on advanced computer architectures. There are two very similar efforts towards running high-resolution climate models on parallel architectures. One of these is the Parallel Climate Model (PCM) at NCAR led by Warren Washington [3]. In this model, each component model is a parallel model, but all of the components are combined into a single executable and run serially in a single partition of a parallel machine. We have instead used an approach very similar to the NCAR CSM model, using a set of autonomous component models running as separate executables and communicating using message-passing through a flux coupler. In the next section, we will describe the model in detail. The following sections will examine aspects of the model which affect computational performance and describe future improvements in the model.

II. Model Description

The Los Alamos Coupled Model utilizes the framework of the NCAR CSM model in which ocean, atmosphere, land and sea ice models are run as separate autonomous executables which communicate through another executable called a flux coupler. We are using the NCAR Flux Coupler [4] with a few modifications which will be described later. The flux coupler keeps all the models synchronized and computes many of the interface fluxes between models. Most component models send their state variables at the atmosphere/ocean interface to the coupler and the coupler computes fluxes based on those state variables; exceptions to this rule will be discussed below. Fluxes computed by the coupler are computed on the finest component model grid (usually the ocean grid), requiring the flux coupler to remap fields from one grid to another. One of the changes we have made to the CSM flux coupler is to include a very general conservative remapping scheme developed at Los Alamos which will perform first and second-order conservative remappings for *any* grid on a sphere [5]. Another issue resulting from the use of different grids is the consistency of land masks. Because the flux coupler computes most fluxes on the ocean grid, the ocean model serves as the "master" grid and all other grids must conform to the land/ocean mask on that grid. In some cases, land mask discrepancies result in the land model computing land values unnecessarily over ocean points and the coupler treats this correctly by simply multiplying that value by a land fraction of zero. However, problems occur when the ocean model dictates that some fraction of an atmosphere grid cell should be land, but the land model treats the cell as an ocean point. This can occur on continental margins or for inland seas that the ocean model ignores. Each of these cases require altering the land-model mask to compute land values at these points.

The flux coupler and component models communicate through the Model Coupling Library (MCL) [6] developed by John Dennis at NCAR. This library provides a flexible message-passing fabric, allowing the user to choose the communication protocol and providing a relatively robust error detection mechanism so models can shut down gracefully if a particular component stops prematurely.

The ocean model is the Los Alamos Parallel Ocean Program (POP) developed by Smith, Dukowicz and Malone [7] based on earlier models by Bryan [8], Cox [9], Semtner [10] and Chervin [11]. The POP model was written specifically for parallel machines and supports a variety of programming models, including message-passing, shared-memory and data-parallel. The model integrates the primitive equations using a B-grid for the horizontal discretization and depth (z) as the vertical coordinate. The primitive equations are split into baroclinic and barotropic modes and the baroclinic modes are advanced in time using a leap-frog scheme. The barotropic equations have been formulated to solve for the surface pressure and the equations with a free-surface boundary are solved implicitly using a preconditioned conjugate gradient solver [12]. For the simulations described here, the Gent-McWilliams parameterization [13] for mixing along isopycnal surfaces is used as well as the k-profile parameterization (KPP) [14] for vertical and mixed-layer mixing. The POP model also uses a displaced-pole grid [15] (see Figure 1) which allows simulation of the arctic regions without the use of filtering or restrictive time steps. This is very useful in coupled climate simulations where arctic processes are extremely important. Our initial ocean resolution is a global grid at an average horizontal resolution of 2/3 degree and 32 vertical levels with non-uniform spacing (finer vertical resolution near the surface). Due to the relatively slow timescales in the ocean, the ocean model communicates with the flux

coupler only once per day. Sea surface temperature, surface salinity, two velocity components and the surface slope are sent to the coupler while the coupler sends to the ocean the wind or ice stress, net shortwave radiation flux, total non-shortwave heat flux (longwave, sensible and latent heat) and total water flux (precipitation, evaporation, melting and river runoff).

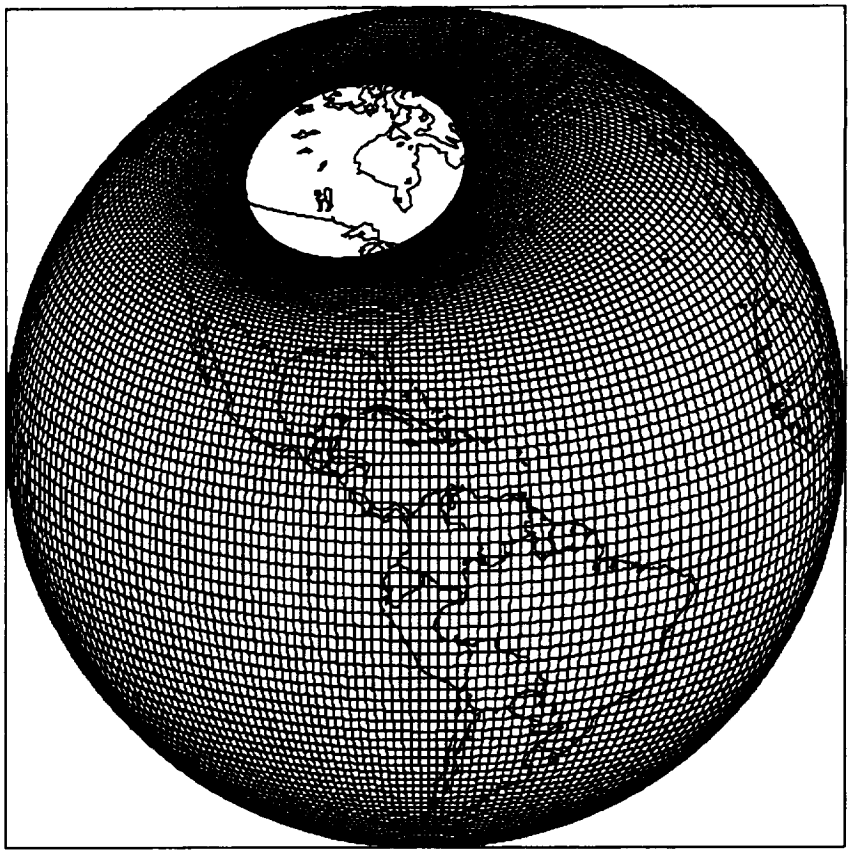


Figure 1. The POP displaced-pole grid with the pole shifted into the North American continent.

Sea ice is simulated by the CICE model, a new ice model developed by Hunke and Dukowicz at Los Alamos. The CICE model utilizes an elastic-viscous-plastic ice rheology [16] for the ice dynamics which allows a fully-explicit formulation ideal for parallel computers. The ice thermodynamics is computed using a three-layer model of Semtner [17,18]. Currently, this model uses a directive-based loop-level parallelism which can be used on shared-memory machines like the SGI/Cray Origin 2000. Modifications for a message-passing version are in progress. The CICE model uses the same generalized grid that the POP ocean model uses and in the model presented here is always run on the same ocean grid, eliminating the need for remapping between ice and ocean. Unlike the other components, the CICE model computes the surface temperature, latent heat, sensible heat and upward longwave flux self-consistently using an iterative method. The ice also

responds relatively rapidly to the winds from the atmosphere and the ice model therefore communicates with the flux coupler every two hours. The ice model sends to the coupler the ice fraction, surface temperature, four albedo components, stress at the ice/ocean interface, heat flux due to melting/freezing, total water flux at the ice/ocean interface, shortwave radiation that penetrates through the ice, latent heat, sensible heat, upward longwave and evaporative water flux. From the coupler, the ice model receives sea surface temperature, salinity, two ocean velocity components, ocean surface slope, conductive heat flux from ocean, height of first atmospheric model level, atmospheric wind speed, atmospheric potential temperature, specific humidity, net shortwave radiation flux, downward longwave radiation flux, wind stress and precipitation water flux. Because CICE computes many fluxes internally and exists at the interface between atmosphere and ocean, it requires exchanging a large number of fields with the flux coupler.

For the work presented here, the atmosphere and land models are combined in the NCAR Community Climate Model (CCM3) [19]. Recently, the land model has been separated from the atmosphere model into the NCAR Land Surface Model (LSM). The CCM3 model is a global spectral model for the atmosphere which is too detailed to describe fully here. The land surface model is a comprehensive physical model of energy, momentum, water and CO_2 exchange between the atmosphere and land with varying soil and vegetation types [20]. The CCM3 model is typically run at T42 resolution (approximately 2.8 degrees) in the horizontal and 18 levels in the vertical. Currently, CCM3 uses directive-based loop-level parallelism with a one-dimensional decomposition in latitude (the outer latitude loop is parallelized). A full message-passing version with two-dimensional decomposition is in progress through a collaboration between NCAR and Oak Ridge National Laboratory. The atmosphere has the shortest timescale variability (in the surface fields) of any model and must therefore communicate the most frequently. Initially, CCM3 communicated every model step (20 minutes) but this frequency was reduced to one hour for performance reasons (see next section). The atmosphere sends to the coupler the height of the first model level, two wind components, potential temperature, specific humidity, density, net shortwave radiation, downward longwave radiation, and precipitation water flux. The atmosphere receives four albedo components, surface temperature at every point, wind stress, latent heat flux, sensible heat flux, upward longwave heat flux and evaporative water flux. The land model, like the ice model, computes many fluxes internally and passes to the flux coupler the four albedo components over land, land surface temperature, latent heat flux, sensible heat flux, upward longwave heat flux and evaporative water flux over land points.

As mentioned above, the flux coupler computes many of the interface fluxes, including the wind stress, latent, sensible, upward longwave heat flux and evaporative water flux over ocean points. It also computes albedoes over the ocean. Because the land model currently has no river runoff model, we have added a simple runoff scheme into the coupler. This scheme computes the excess evaporation over precipitation on ocean grid points, representing the net excess of precipitation over land area. This excess is distributed at river outlets with a weighting based on annual-average river output. Lastly, the coupler contains routines for averaging the various fields before sending the fields to the appropriate model. The averaging can be in space when combining land/ocean values before sending to the atmosphere (based on the fractional area of the cell covered by land, ocean and ice). Averaging in time occurs for models which communicate less frequently; the ocean for example is sent the average fluxes over the previous day.

III. Performance Issues

Obviously, the primary factor that affects performance of the coupled model is the performance of each individual component. However, running each component as separate executables offers another level of parallelism which can be exploited. Inter-model parallelism is not immune to factors which interfere with efficient parallelism in individual component models. One impediment to parallelism occurs when the interaction between models results in serial dependencies. Efficient parallelism is also inhibited when the components are not well load-balanced, resulting in processors assigned to a component remaining idle because that component must wait for another component to finish.

Serial dependencies result from the choices of which model computes fluxes and how often those fluxes must be exchanged. For example, the flux coupler is computing the upward longwave, latent and sensible heat fluxes. If the atmosphere is to respond immediately to these fluxes, it must communicate every time step (20 minutes for CCM3 at T42 resolution). However, this means that the coupler and the atmosphere model are running serially because each must wait until the other finishes before continuing. If the coupler is running sufficiently fast on very few processors this is not a problem, but we have found that in practice this would require allocating too many processors to the coupler which would then sit idle while waiting for the atmosphere. Instead, the atmosphere model can communicate less frequently (once per hour) and can integrate for three time steps before having to stop and exchange messages with the coupler. The serial interaction with the coupler is then a much smaller fraction of the total running time.

The second reason for serial dependencies is that after a model receives a message from the coupler, it may need to perform some calculations before sending a message back to the coupler. The coupler may be waiting for this information before it can continue so it is important to minimize the amount of work between a receive and a send in each component model. In some models, the work between messages can be eliminated by using variables from a previous time level. For example, the ocean communicates with the coupler once per day, receiving fluxes from the coupler that have been averaged over the previous day and sending back to the flux coupler ocean state variables from the previous time step. In this case, no work needs to be performed since the necessary information is readily available and can be sent immediately after a receive. This is acceptable for the ocean model because the ocean responds slowly in comparison to the other models. Unlike the ocean model, the ice model responds more quickly to fluxes like the wind stress. In addition, the ice model computes surface latent and sensible heat fluxes internally which are needed immediately by the atmosphere model. When the CICE model was first coupled with the atmosphere and ocean, the model was written in such a way that it would receive information from the coupler and advance an entire time step before sending the necessary flux information back to the coupler. However, the fluxes required were actually computed first in the relatively fast thermodynamics phase, so we could reduce the serial dependence greatly by sending the fluxes back immediately after the thermodynamics phase and the ice dynamics could then proceed in parallel with the other components.

In order to use computational resources most efficiently, it is necessary to load balance the component models so that processors associated with a component are not wasted idly waiting for another component to finish before they can synchronize. The load balancing process is made easier if the scaling of each model is predictable and if the model is flexible enough to run on any configuration of processors. Figure 2 shows the scaling of each

component model running on a Silicon Graphics/Cray Research Origin 2000. The results indicate that directive-based parallelism for multi-threading at the loop level does not appear to scale well above 16 processors. Because we would like to use as many processors as possible to integrate in the shortest time, this poor scaling will create problems. Currently, for a load-balanced run, we would require 32 processors for the atmosphere, 16 processors for the ocean and four processors for CICE. Trying to run at higher processor counts would not improve the total run time because CCM3 would not run as efficiently at 64 processors. The current version of CCM3 also will not scale beyond 64 processors at T42 resolution because the parallelization only occurs over the latitude loop. As mentioned above, a version of CCM3 with message-passing and a two-dimensional decomposition is in progress and this should allow us to run much more efficiently at higher processor numbers. Another example of poor scaling in Figure 2 is the conservative remapping scheme. One portion of this scheme which accumulates partial sums scales very poorly with processor number. This non-scaling part of the routine will also dominate as we move toward higher resolution models, so while the run time is currently relatively small for this combination of models, the remappings could create problems at higher resolutions. We are currently investigating more efficient methods for implementing these remappings.

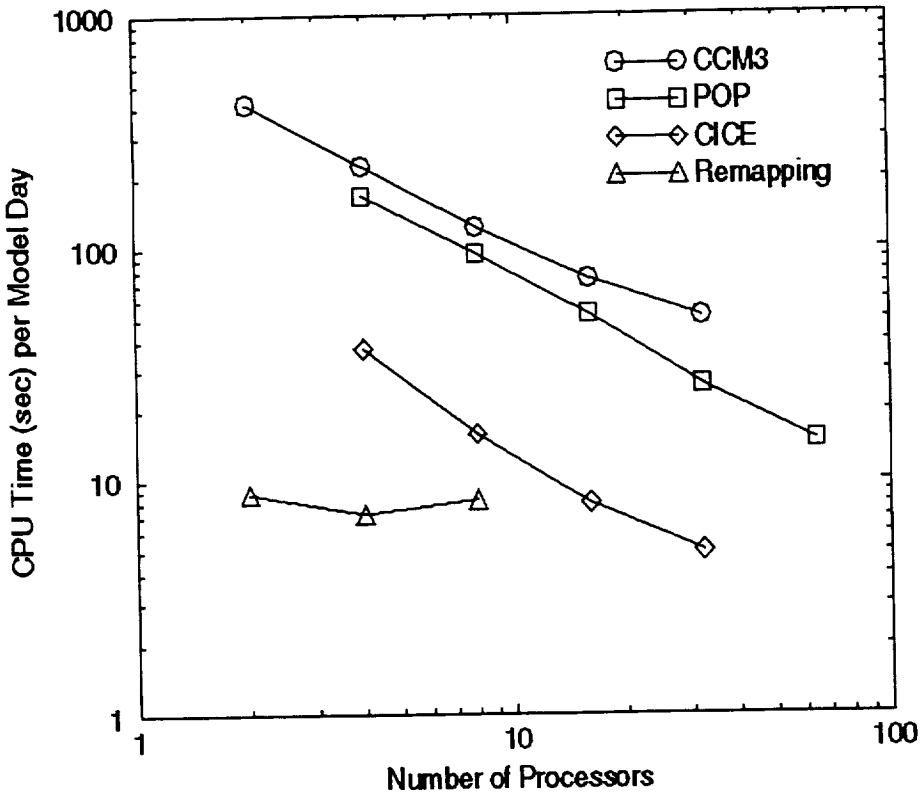


Figure 2. CPU time per model day for component models as a function of the number or processors. CCM3 is at T42 resolution while POP and CICE are at 2/3 degree resolution. The remapping times are the time spent in the coupler remapping fields from one grid to another.

As mentioned above, we have used the Origin 2000 computer for our simulations. The Origin 2000 is a distributed-shared-memory machine, meaning it has a cache-coherent globally-addressible memory which is physically distributed across processors. This architecture has the advantage that it supports a variety of programming models so that we can mix message-passing codes with multi-threaded codes. We have taken full advantage of this feature of the architecture. The current implementation of distributed-shared-memory also has some disadvantages. Because all processors can access memory across the machine, the operating system currently can not ensure that processors and memory are truly dedicated to a particular model. This can create problems if one of the threads from the atmosphere, ice or coupler begins to use the memory of a processor running another component model, resulting in a substantial degradation in performance for both processes involved. In practice, such a situation can be avoided if all users of a machine give the batch scheduler the proper resource parameters (number of processors and memory size) so that the scheduler does not oversubscribe the resources of the machine. It is also important to know that the system spawns an extra process for parallel jobs so that when running a 32-processor MPI job, the user should ask for 33 processors.

Our system at Los Alamos is actually a cluster of Origin 2000 machines. The simulations we have run so far have only utilized single boxes of up to 128 processors. The MCL communications library is flexible enough to run each component on a separate box. The batch scheduler also can allocate processors across boxes in any particular configuration, but currently cannot allocate particular executables and their job scripts to run simultaneously on the properly-sized set of processors. This capability will be necessary as we move to very high resolutions.

IV. Conclusions

We have found the NCAR Flux Coupler concept to be a very flexible and efficient way for us to quickly couple the POP ocean model, the CICE ice model and the CCM3 atmosphere and land model. We continue to improve the performance of each of the components in order to increase the efficiency of the model. In particular, we are working to improve the remappings within the flux coupler in order to reduce the amount of time spent in that component. Additionally, we are working on improvements to the POP model that could substantially improve the performance on cache-based microprocessor and allow us to run at higher resolutions.

The flux coupler concept may present problems for other model combinations or computer architectures. For example, in the coupled model described above, the CICE component requires the upward longwave, latent and sensible heats to be computed implicitly and self-consistently with the surface temperature. If we were to couple POP and CICE to an atmosphere model which also required these fluxes to be computed internally in the atmosphere model, one of the models would have to be altered to accept fluxes computed externally. Computer architecture flexibility is also required if all the components use different programming paradigms. Distributed-shared-memory machines are ideally suited as they can be programmed using message-passing, multi-threading or shared-memory paradigms. Other architectures have operating systems which do not allow separate executables to run simultaneously in the same partition.

Acknowlegments. This work of course would not be possible without the help of the developers of each component model, including Elizabeth Hunke, Mat Maltrud, Rick Smith, John Dukowicz, John Dennis, Dave Williamson, and Jim Hack. The work was performed using facilities of the Advanced Computing Laboratory and was supported by the Department of Energy's Climate Change Prediction Program.

References

1. R.J. Stouffer, S. Manabe and K. Bryan, Interhemispheric asymmetry in climate response to a gradual increase of atmospheric CO_2, *Nature*, **342**, 660 (1989).

2. B.A. Boville and P.R. Gent, The NCAR Climate System Model, version one, *J. Climate*, in press (1998).

3. http://www.cgd.ucar.edu/ccr/pcm

4. F.O. Bryan, B.G. Kauffman, W.G. Large and P.R. Gent, The NCAR CSM Flux Coupler, *NCAR Technical Note No. 424* , National Center for Atmospheric Research, Boulder, CO (1996).

5. P.W. Jones, First and second-order conservative remapping schemes for grids in spherical coordinates, in preparation (1998).

6. J. Dennis, The glue that holds it together: NCAR's model coupling library, to appear in *Making Its Mark: The Use of Parallel Processors in Meteorology*, ed.G-R. Hoffman (1998).

7. R.D. Smith, J.K. Dukowicz and R.C. Malone, Parallel ocean general circulation modeling, *Physica D*, **60**, 38 (1992).

8. K. Bryan, A numerical method for the study of the circulation of the world ocean, *J. Comp. Phys.*, **4**, 347 (1969).

9. M.D. Cox, A primitive-equation three-dimensional model of the ocean, *GFDL Ocean Group Tech. Rep. No. 1*, GFDL/NOAA, Princeton Univ., Princeton, NJ (1984).

10. A.J. Semtner, Jr., Finite-difference formulation of a world ocean model, in *Advanced Physical Oceanographic Numerical Modeling,* ed. J.J. O'Brien (Dordrecht: Reidel, 1986).

11. R.M. Chervin and A.J. Semtner Jr., An ocean modelling system for supercomputer architectures of the 1990s, in *Proc. of the NATO Advanced Research Workshop on Climate-Ocean Interaction*, ed. M. Schlesinger (Dordrecht: Kluwer, 1988).

12. J.K. Dukowicz and R.D. Smith, Implicit free-surface method for the Bryan-Cox-Semtner Ocean Model, *J. Geophys. Res.*, **99**, 7991 (1994).

13. P.R. Gent and J.C. McWilliams, Isopycnal mixing in ocean circulation models, *J. Phys. Oceanogr.*, **20**, 150 (1990).

14. W.G. Large, J.C. McWilliams and S.C. Doney, Oceanic vertical mixing: a review and a model with a nonlocal boundary layer parameterization, *Rev. Geophys.*, **32**, 363 (1994).

15. R.D. Smith, S. Kortas and B. Melz, Curvilinear coordinates for global ocean models, *J. Comp. Phys.*, submitted.

16. E.C. Hunke and J.K. Dukowicz, An elastic-viscous-plastic model for sea ice dynamics, *J. Phys. Oceanogr.*, **27**, 1849 (1997).

17. A.J. Semtner Jr., A model for the thermodynamic growth of sea ice in numerical investigations of climate, *J. Phys. Oceanogr.*, **6**, 379 (1976).

18. G.A. Maykut and N. Untersteiner, Some results from a time dependent thermodynamic model of sea ice, *J. Geophys. Res.*, **76**, 1550 (1971).

19. J.T. Kiehl, J.J. Hack, G.B. Bonan, B.A. Boville, B.P. Briegleb, D.L. Williamson and P.J. Rasch, Description of the NCAR Community Climate Model (CCM3), *NCAR Technical Note NCAR/TN-420+STR*, National Center for Atmospheric Research, Boulder, CO (1996).

20. G.B. Bonan, The land surface climatology of the NCAR land surface model coupled to the NCAR Community Climate Model (CCM3), *J. Climate*, in press (1998).

Author: Rodney James
National Center for Atmospheric Research
P.O. Box 3000
Boulder, CO 80307
rodney@ncar.ucar.edu

Co-author(s): T. Bettge
S. Hammond

Portability and Performance of a Parallel Coupled Climate Model

The Parallel Climate Model (PCM) is a coupled climate system model being developed in a collaborative effort at NCAR with support from the DOE Climate Change Prediction Program (which includes scientists and software engineers from the Los Alamos National Laboratory and the Naval Postgraduate School). PCM has four components: ocean, atmosphere/land, and ice models, and a flux coupler, all combined into a single portable program which can be run on several parallel platforms. The ocean, atmosphere/land, and ice component models are configured on different grids with different time steps. Component model interactions, both temporal and spatial, are coordinated through the flux coupler. In this paper, we discuss the details of the coupled model software architecture and the strategies used for maintaining both code portability and performance portability across platforms. We present details of the computational and communication efficiency for the complete coupled model as well as the individual components on different systems from different vendors. Finally, we compare and contrast PCM performance and scaling results on several different parallel machines for four processors up to hundreds of processors.

An atmospheric general circulation model with chemistry for the CRAY T3E: Design, performance optimization and coupling to an ocean model

John D. Farrara, Leroy A. Drummond, Carlos R. Mechoso, Joseph A. Spahr
Department of Atmospheric Sciences, University of California, Los Angeles, Mail Stop 156506, Los Angeles, CA 90095-1565, jfarrara@ucla.edu, +1 310 825-9205, FAX: +1 310 206-5219

Tom Clune
SGI/Cray Research

Abstract

The design, implementation and performance optimization on the CRAY T3E of an atmospheric general circulation model (AGCM) which includes the transport of, and chemical reactions among, an arbitrary number of constituents is reviewed. The parallel implementation is based on a two-dimensional (longitude and latitude) data domain decomposition. Initial optimization efforts centered on minimizing the impact of substantial static and weakly-dynamic load imbalances among processors through load redistribution schemes. Recent optimization efforts have centered on single-node optimization. Strategies employed include loop unrolling, both manually and through the compiler, the use of an optimized assembler-code library for special function calls, and restructuring of parts of the code to improve data locality. Data exchanges and synchronizations involved in coupling different data-distributed models can account for a significant fraction of the running time. Therefore, the required scattering and gathering of data must be optimized. In systems such as the T3E, there is much more aggregate bandwidth in the total system than in any particular processor. This suggests a distributed design. The design and implementation of a such distributed 'Data Broker' as a means to efficiently couple the components of our climate system model is described.

1. Introduction

The primary means of studying the Earth's climate and its variability is numerical modeling. Numerical models of the climate system must account for the complex interactions and feedbacks among its components. Examples of phenomena that can be studied using such 'climate system models' are El Niño-Southern Oscillation events, the role of the oceans in moderating the greenhouse warming effect of carbon dioxide and other gases, and the behavior of the ozone layer. The primary components of the climate system are the atmosphere and ocean, which are represented by general circulation models (GCMs). Led by Akio Arakawa, the Department of Atmospheric Sciences at UCLA has pioneered the development of atmospheric GCMs (AGCMs). These models are constantly being improved through revisions in their numerical schemes and physical parameterizations as well as through the incorporation of new physical processes. Simulations using GCMs are computationally very demanding because there are a large number of physical quantities to be predicted and also because very long simulations (on the order of 10^7 model timesteps) are usually required. We are now entering "the great challenge" third phase of

atmospheric modeling (Arakawa [1]). To meet the challenges of this phase, it is essential to: 1) revise the dynamical core of atmospheric models and adjust the physical parameterizations to accommodate the new core, and 2) optimize the model's code for high performance computing environments. In this paper, we address the second of these required developments with an account of the design, optimization and performance of the UCLA GCM code including chemical tracers in scaleable parallel computing environments. We also review the design and implementation of a system we have developed to efficiently couple this model to an ocean GCM.

2. The AGCM and the parallel implementation of its code

The UCLA AGCM is a state-of-the-art finite-difference model of the global atmosphere. The model predicts the horizontal wind, potential temperature, water vapor mixing ratio, planetary boundary layer depth and the surface pressure as well as the surface temperature and snow depth over land. The model also optionally predicts the concentrations of an arbitrary number of chemical tracers. The horizontal finite-differencing is done on a staggered Arakawa "C"-grid which is regular in longitude and latitude and is a fourth-order accurate version of the differencing scheme of Arakawa and Lamb [2]. The differencing of the thermodynamic energy and water vapor (and other tracers) advection equation is also based on a fourth-order accurate scheme. The vertical coordinate used is the modified sigma coordinate of Suarez et al. [3]. In this coordinate, the lowest model layer is the planetary boundary layer. The vertical finite differencing is performed on a Lorenz-type grid following Arakawa and Lamb [2] in the stratosphere and Arakawa and Suarez [4] in the troposphere. This differencing is of second order accuracy and is designed to conserve the global mass integrals of potential temperature and total energy for adiabatic, frictionless flows. For the integration in time of the momentum, thermodynamic energy and tracer advection equations, a leapfrog time differencing scheme is used with a Matsuno step regularly inserted to prevent separation of the solution. Filtering of selected terms in the prognostic equations is performed at high latitudes (see Section 2b). A nonlinear horizontal diffusion of momentum, with a small coefficient, is included following Smagorinsky [5]. This diffusion is applied at each time step, using a forward time differencing. In layers where an unstable stratification of mass develops (potential temperature decreases with height), it is assumed that sub-grid scale dry convection occurs and the prognostic variables in the layers involved are completely mixed.

Planetary boundary layer processes are parameterized using the mixed-layer approach of Suarez et al. [3]. In this parameterization, surface fluxes are calculated following the bulk formula proposed by Deardorff [6]. Parameterization of the effects of cumulus convection, including its interaction with the PBL, follows Arakawa and Schubert [7] and Lord et al. [8], with a relaxed adjustment time scale as described in Ma et al. [9] (see Section 2c). The parameterization of both terrestrial and solar radiative heating follows Harshvardhan et al. [10, 11]. The cloud optical properties are specified as in Harshvardhan et al. [11]. This prescription makes a distinction between stratiform and "cumulus anvil"-type clouds. "Cumulus anvil"-type clouds are assumed to exist at each model layer above 400 mb where the cumulus mass flux is positive; stratiform clouds are assumed to occur at grid points where the predicted relative humidity exceeds 100%. The effects of subgrid-scale orography are included via a gravity wave drag parameterization (Kim and Arakawa [12], Kim [13]). The transformations of chemically active gases and aerosol tracers in the atmosphere are described by the UCLA atmospheric chemistry model (ACM). This model includes algorithms to solve photochemical and thermochemical coupled systems, a detailed treatment of the microphysics of small particles, and a fully integrated radiation package (Elliot et al. [14] and references therein). The geographical distribution of sea ice and sea surface temperatures are prescribed on a monthly basis. Surface albedo and roughness lengths are specified following Dorman and Sellers [15], in which roughness lengths over land vary according to vegetation type.

Daily values of these surface conditions are determined from the monthly mean values by linear interpolation.

a. Structure and parallel implementation of the code

In AGCMs physical processes are modeled either explicitly or implicitly via parameterization of the effects of subgrid physical processes on the grid-scale processes. This division is reflected in the UCLA AGCM code in that there are two major code components: AGCM/Dynamics, which computes the evolution of the fluid flow governed by the primitive equations, and AGCM/Physics, which computes the effects of subgrid scale processes (such as cumulus convection) on the grid scale (see Figure 1). Included in the AGCM/Physics are the calculations performed by the ACM. The results obtained by AGCM/Physics are supplied to AGCM/Dynamics as forcing for the flow calculations. The parallel version of the UCLA AGCM code was designed for distributed memory multiprocessor computing environments (Wehner et al. [16]). A two-dimensional domain decomposition of the horizontal data domain (longitude-latitude) is used. Subdomains thus consist of a set of vertical columns from the Earth's surface to the top of the atmosphere. This choice is based on the fact that many physical processes are strongly coupled in the vertical which makes parallelization less efficient in that direction. In addition, the number of grid points in the vertical is usually small compared to the number in the horizontal. There are two types of interprocessor communications required during the simulation: 1) data exchanges among (logically) neighboring processors to accomplish the finite differencing, 2) data exchanges among remote processors necessary for implementation of the high latitude Fourier filtering (see Section 2b). We next review the structure and implementation of the most computationally intensive algorithms, first those in the AGCM/Dynamics, then those in the AGCM/Physics.

b. AGCM/Dynamics kernels

i) Vertical discretization

The model is formulated using a modified sigma (pressure divided by surface pressure) coordinate in which the top of the model layer nearest the Earth's surface (called the planetary boundary layer, or PBL) is a coordinate surface (Suarez et al. [3]). Hence, the pressure levels corresponding to particular values of sigma vary with time. The prognostic variables are staggered in the vertical following the Lorenz scheme, in which potential temperature, the horizontal wind components and the mixing ratio of water vapor are defined in the middle of the layers, while the diagnostically determined vertical velocity is defined at the interfaces between the layers. Accomplishing this vertical differencing is computationally demanding because it requires the calculation at every timestep of several functions of pressure involving exponentials and real numbers raised to a real power. We describe our approach to optimizing this part of the code in Section 3a.

ii) Polar filtering

To avoid use of the extremely small timestep necessary to satisfy the Courant-Friedrich-Levy (CFL) condition near the poles, a longitudinal averaging (which takes the form of a Fourier filter) is performed on selected terms in the momentum equation. This filter acts poleward of 45° latitude and its strength is gradually increased towards the pole by increasing the number of affected zonal wavenumbers and the amount by which they are damped (Arakawa and Lamb [4]). As explained in Lou and Farrara [17], the high cost of performing this filtering stems from two factors. The first is the substantial non-nearest neighbor communication required. The second is the severe load imbalance imposed by the fact that only subdomains containing high-latitude points perform this filtering. Lou and Farrara [17] describe a load-balancing scheme for the filtering

which is based on evenly distributing complete longitude-height slices of fields to be filtered among all processors. We describe our attempts to further optimize this part of the code in Section 3b.

c. AGCM/Physics kernels

i) Terrestrial radiation

The parameterization of terrestrial radiation follows Harshvardhan et al. [10, 11]. This scheme includes the effects on longwave radiative transfer of water vapor, carbon dioxide and ozone as well as convective and stratiform clouds. The absorption/emission of radiation is determined using the broadband transmission approach of Chou [18] for water vapor, that of Chou and Peng [19] for carbon dioxide and that of Rodgers [20] for ozone. In this approach, the transmission functions for reference atmospheric temperature profiles at specified pressure levels are precomputed using highly accurate line-by-line methods. Adjustments are then made to these precomputed transmission functions via 'scaling functions', together with the temperature deviations from the reference profile, to obtain transmission functions for other atmospheric profiles. Thus, the success of the parameterization rests in part on the choice of reference conditions, and these are chosen to correspond to the regions of peak cooling in the atmosphere. Transmittances are computed in five bands, two each for water vapor and carbon dioxide and one for ozone. The original implementation of this code [18] was designed for vector supercomputers and performed table look-ups of the precomputed transmission functions. The table lookups were replaced by fitting algebraic functions to the tables since the table lookups were difficult to efficiently vectorize. The algebraic functions are mostly exponentials, while the scaling functions contain the scaled pressure raised to a real constant. In order that the code vectorize efficiently it was structured to perform calculations on multiple atmospheric columns simultaneously. However, this structure is not optimal for the T3E processors since the cache reuse is poor. We describe our restructuring of this code for greater efficiency on cache-based architectures in Section 3c.

ii) Cumulus convection

The Arakawa-Schubert parameterization (Arakawa and Schubert [7], Lord et al. [8]) is used to compute the interactions of grid-scale prognostic variables and subgrid-scale cumulus convection. Specifically this parameterization determines the mass fluxes produced by subensembles of cumulus clouds originating in the PBL. These mass fluxes are used to obtain the heating, moisture source/sink and momentum redistribution in the vertical. Multiple cloud types are permitted and all clouds of a particular type are assumed to be identical. The total collection of cumulus clouds is referred to as a cumulus ensemble, and the clouds that belong to a particular cloud type are referred to as a subensemble. Each subensemble is distinguished from the others by its fractional entrainment rate, which is assumed to be independent of height.

There are two primary components to the AS scheme. The first component describes how the internal sounding of an individual cumulus cloud is controlled by the large-scale environment in which it develops. This component is called the "static control". The second component describes how an ensemble of cumulus clouds modifies the large-scale thermodynamic structure of the atmosphere. This component of the parameterization is called the "feedback". The large-scale tendencies due to a particular subensemble are proportional to the cumulus mass flux at the cloud base (the PBL top) for that subensemble. The AS parameterization can thus be closed by determining cumulus mass fluxes at the cloud base associated with each subensemble. This closure is achieved through a description of how the cloud-base mass fluxes are controlled by the

evolution of the large-scale environment. This is called "dynamic control". Specifically, the rate at which conditional instability is generated by large-scale processes is very nearly balanced by the rate at which cumulus clouds suppress conditional instability by their feedback on the large-scale environment. This is referred to as the "quasi-equilibrium" hypothesis. As a measure of conditional instability, AS defined a cloud work function (CWF) as the rate of generation of cumulus cloud-scale kinetic energy for a particular subensemble, per unit cloud base mass flux into that subensemble. According to this definition, the CWF as well as the cloud base mass flux are positive or zero. Positive values of the CWF indicate the existence of conditional instability for a given subensemble. The CWF is computed from the environmental sounding using a simple cloud model. In practice, solving for a set of cloud base mass fluxes (one for each subensemble) subject to the condition that all must be non-negative is mathematically tricky and expensive computationally.

Since the model has a discrete vertical structure, cloud types (subensembles) are denoted by their cloud-top pressure rather than fractional entrainment rate. However, since the fractional entrainment rate is needed to determine the in-cloud sounding, it is necessary to solve for the fractional entrainment rate that is consistent with each cloud-type each time the parameterization is invoked. Since this solution must be obtained iteratively, it is rather expensive computationally. The rate at which conditional instability is generated by large-scale processes is straightforwardly determined. The rate at which each cloud type suppresses conditional instability (and thereby interacts with every other cloud-type) is somewhat more complicated to compute. A small "trial mass flux" is assigned to each cloud type in turn. This trial mass flux is assumed to act over a time interval on which the parameterization is called, producing changes in the temperature and moisture profiles through the feedback process; the implied change in each cloud-type's work function is then evaluated. In Section 3d we explore a computationally less expensive option to this algorithm.

3. Optimization of major computational kernels

In this section, we discuss the results of our effort to optimize the most computationally intensive algorithms, first those in the AGCM/Dynamics, then those in the AGCM/Physics. It should be noted that there are a number of other routines (parts of the algorithm) that contribute significantly to the overall execution time; this makes the optimization of GCMs especially challenging. All timings reported in this section correspond to the wall-clock times required to simulate one day using the 2.5^{o} longitude, 2^{o} latitude, 29 layer version of the model running on a CRAY T3E-600 with 2 chemically active species (CFCs). In each of the Tables below we present (in addition to the timings for particular kernels) the total model execution time; note that this time succesively decreases as each of the optimizations is included. The starting point for these total times (125 seconds/simulated day) corresponds to a version of the code that uses: i) the original implementation of the vertical discretization code, the load balanced filtering code of Lou and Farrara [17], the multi-column implementation of the Harshvardhan [10, 11] longwave radiation code (including calls to assembler coded special functions) and the standard Arakawa-Schubert [7, 8] cumulus convection parameterization.

a. AGCM/Dynamics kernels

i) Vertical discretization

The vertical differencing is computationally demanding because it requires the calculation at every timestep of several functions of pressure involving exponentials and real numbers raised to a real power. To reduce the impact of computing these expensive special functions we are using vectorized assembly-coded routines to perform these operations (Drummond et al [22]). The

reduction in execution time obtained is given in Table 1. The impact on the time spent in the vertical differencing is substantial (40% reduction), but the overall run time is only marginally improved.

Table 1. Vertical Discretization Time (seconds/simulated day).

	Original Implementation	With assembler-coded special functions
Vertical discretization time	5.25	3.12
Total model execution time	125	123

ii) Polar filtering

Lou and Farrara (1996) describe a load-balancing scheme for the filtering which is based on evenly distributing complete longitude-height slices of fields to be filtered among all processors. When large numbers of processors are used the load redistribution with this method becomes poor as the number of processors can be nearly as large or larger than the number of slices to be filtered. Therefore, we have modified this scheme to break up these longitude-height slices and redistribute individual longitudes (rows) of data. This results in a larger number of units of work to be redistributed, giving a better load balance on large numbers of processors. This can be seen in Table 2 which give the timings of the original load-balanced filtering and the revised load-balanced filtering algorithms. The time spent in filtering has been reduced by 40% resulting in a 10% reduction in total execution time.

Table 2. Total Polar Filtering Time (seconds/simulated day).

	Original Load Balanced Filtering [17]	Revised Load Balanced Filtering
Filter time	20.5	12.3
Total model execution time	123	114

b. AGCM/Physics kernels

i) Terrestrial Radiation

The exponential and real raised to a real power operations, as in the vertical discretization, are the most expensive operations in the terrestrial radiation. As in the vertical discretization code, we are using vectorized assembler-coded routines to perform these operations. As indicated in Section 2c, this code was originally structured for vector supercomputers, such that calculations were performed on multiple atmospheric columns simultaneously. On machines such as the T3E, the overhead in processing a set of instructions from a program is contributed primarily by the retrieval of data from main memory into cache. In an attempt to minimize the traffic of data between cache and main memory, we have re-written the terrestrial radiation code to operate on only a single atmospheric column at a time, thereby reducing the sizes of many of the arrays used in this part of the code and increasing substantially the cache reuse. The timing for the original and restructured versions (assembler routine calls for special functions are used in *both* cases) of this code are given in Table 3. Table 3 shows a reduction in time spent in the terrestrial radiation computation of approximately 20% as well as a modest reduction in the overall time of about 4%.

Table 3. Terrestrial Radiation Time (seconds/simulated day).

	Multi-column (original) Implementation	Single-column Implementation
Terrestrial Radiation time	21.9	17.1
Total model execution time	114	110

ii) Cumulus convection

There are significant mathematical and computational drawbacks to the standard Arakawa-Schubert cumulus parameterization we have been using (see Section 2). Therefore, we decided to replace it with a revised version of the algorithm called "prognostic" Arakawa-Schubert (Randall and Pan [21]). In this "prognostic" version, the CWF quasi-equilibrium is relaxed by predicting the cloud-scale kinetic energy. It has been shown (Randall and Pan [21]) that this prognostic version reduces in principle to the CWF quasi-equilibrium as the dissipation time scale for the cloud kinetic energy goes to zero. In practice, it has been found that the time-averaged cloud base mass flux, and, therefore, time-averaged cumulus heating and drying are approximately the same as that obtained with the standard AS parameterization. One of the main advantages of this approach is that the difficulties and expense of simultaneously diagnosing a set of physically reasonable cloud base mass fluxes that satisfy the CWF quasi-equilibrium are avoided. In this scheme, the predicted CKE is used to simply determine the cloud base mass fluxes for each subensemble. A secondary cause for the poor performance of the standard AS code is poor cache reuse; this problem is related to the large number of arrays required to solve for the cloud base mass fluxes and becomes acute as the number of vertical levels increases. This problem is also alleviated in the prognostic version as the code is considerably simplified and the number of arrays is substantially reduced. The impact on performance is huge (see Table 4); the prognostic AS is 6.5 times faster than the standard, yielding a reduction in total execution time of 30%.

Table 4. Cumulus convection Time (seconds/simulated day).

	Standard Arakawa-Schubert	Prognostic Arakawa-Schubert
Cumulus Time	39.8	6.1
Total model execution time	110	76.5

c. *Overall improvement*

The overall reduction in execution time is 48.5 seconds per simulated day. The majority of this improvement (33 seconds) was due to the use of the prognostic version of the Arakawa-Schubert cumulus parameterization. However, the cumulative effect was the other optimizations was significant, amounting to a reduction of more than 1.5 hours in the time required to simulate one year (9.3 hrs --> 7.7 hrs). In addition to the above optimizations, we have found that using the "streams" option on the T3E reduces the overall wall-clock time by additional 15%. However, a hardware bug on the T3E-600 renders the machine unstable when this option is used in conjunction with certain types of calls to the SHMEM library routines for transferring data among processors. Therefore all the timings reported above are for runs that do *not* use this option. Currently, the model code runs 4.4 times faster on the T3E than on the T3D, achieving a peak performance of 35 Gflops on 512 T3E-600 nodes (see Fig. 2). On the T3E, the model code executes approximately 8.5 times faster on 256 nodes than on 16 nodes (the smallest number of

nodes that will run this size problem), resulting in a parallel efficiency of 53%. For this problem size, the parallel efficiency drops off above 256 nodes. Clearly, higher efficiencies can be achieved for higher resolution configurations. Our future work will center on optimizing the ACM portion of the code as this part of the code becomes absolutely dominant when, for example, a 25 species configuration appropriate for the study of stratospheric ozone is used (see Figure 3).

4. Coupling to an ocean GCM

As indicated in Section 2a, the AGCM consists of two major components, AGCM/Dynamics and AGCM/Physics. The ocean general circulation model (OGCM) also has two major components: OGCM/Baroclinic, which determines the deviations from the vertically averaged velocity, temperature and salinity fields, and OGCM/Barotropic, which determines the vertically averaged distributions of those fields. When run on a single node, the AGCM and OGCM codes execute sequentially and exchange fields corresponding to the air-sea interface. The AGCM is first integrated for a fixed period of time and then transfers the time-averaged surface wind stress, heat and water fluxes to the OGCM. This component is then integrated for the same period of time and transfers the sea surface temperature to the AGCM. When run on multiple processors, a scheme that allows the two codes to run in parallel is used. Because AGCM/Dynamics does not exchange data with the OGCM, these components can run in parallel. Further, AGCM/Physics can start as soon as OGCM/Baroclinic completes its calculation, because this includes the sea surface temperature, and can thus run in parallel with OGCM/Barotropic. The efficiency of this scheme depends primarily on having a good balance between the run times for the components running in parallel.

To optimize the required gathering and scattering of fields between different data distributed models we have designed a distributed Data Broker. The Data Broker is designed such that the problem of efficiently coupling model components is been broken up into a small number of relatively independent and reusable kernels, consisting of a Registration Broker, model specific Interpolation routines, and model communications libraries (see Figure 2). The Registration Broker keeps track of the production of, and requests for, multi-dimensional data and their frequencies of production/consumption. The distributed producers and consumers communicate with the Data Broker via the model communications library, which in turn arranges for the appropriate pieces to be transmitted by each of the distributed 'producers' of the fields to each of the distributed 'consumers'. In contrast to the code of the component models, which is 99% FORTRAN, the Data Broker is implemented using C++ and tcl/tk with a FORTRAN-callable interface.

5. Summary

Computer simulations using GCMs are indispensible in studies of the fundamental issues that affect our environment. Such simulations are very demanding of computer resources and eptomize the challenge of Earth Sciences to computer technology. We have addressed this challenge by developing a coupled atmosphere/ocean/chemistry model that makes efficient use of one of today's highest performing computing environments (the CRAY T3E). One of the scientific issues we plan to address with the coupled AGCM/OGCM configuration of this model is the decadal modulation of El Niño-Southern Oscillation events. For this investigation, we require coupled simulations at least several decades or even a century long. To date, available resources have limited us to multi-decadal simulations using a medium range of resolution (for example, the AGCM resolution was coarser than that timed here). With the newly optimized versions of the

model codes, we plan to run century-long coupled simulations using higher resolution for both the AGCM and OGCM. When the results of our current optimization work targeting the ACM are complete we plan investigations of the stratospheric ozone layer. For this, we will perform multi-decadal simulations with the AGCM/ACM at the same high resolution and using the 25 chemical species required to accurately simulate the evolution of ozone in the stratosphere.

Acknowledgments. This work has been supported by the NASA High Performance Computing and Communications for Earth Sciences Project under CAN 21425/041 and the Department of Energy's CHAMMP Program under Grant DE-F03091ER1214.

References

[1] Arakawa, A., 1998: A personal perspective on the early years of general circulation modeling at UCLA. Proceedings, AA fest, January 1998, Los Angeles, CA, in press.

[2] Arakawa, A., and V. R. Lamb, 1977: Computational design of the basic dynamical processes of the UCLA general circulation model. *Methods in Computational Physics,* **17**, Academic Press, New York, 173-265.

[3] Suarez, M. J., A. Arakawa, and D. A. Randall, 1983: The parameterization of the planetary boundary layer in the UCLA general circulation model: Formulation and results. *Mon. Wea. Rev.*, **111**, 2224-2243.

[4] Arakawa, A., and M. J. Suarez, 1983: Vertical differencing of the primitive equations in sigma coordinates. *Mon. Wea. Rev.*, **111**, 34-45.

[5] Smagorinsky, J., 1963: General circulation experiments with the primitive equations. I. The basic experiment. *Mon. Wea. Rev.*, **91**, 99-164.

[6] Deardorff, J. W., 1972: Parameterization of the planetary boundary layer for use in general circulation models. *Mon. Wea. Rev.*, **100**, 93-106.

[7] Arakawa, A., and W. H. Schubert, 1974: Interaction of a cumulus cloud ensemble with the large-scale environment. Part I. *J. Atmos. Sci.*, **31**, 674-701.

[8] Lord, S. J., W. C. Chao, and A. Arakawa, 1982: Interaction of a cumulus cloud ensemble with the large-scale environment. Part IV: The discrete model. *J. Atmos. Sci.*, **39**, 104-113.

[9] Ma, C.-C., C. R. Mechoso, A. Arakawa and J. D. Farrara, 1994: Sensitivity of a coupled ocean-atmosphere model to physical parameterizations. *J. Climate,* **7**, 1883-1896.

[10] Harshvardhan, R. Davies, D. A. Randall, T. G. Corsetti, 1987: A fast radiation parameterization for atmospheric circulation models. *J. Geophys. Res.*, **92**, 1009-1016.

[11] Harshvardhan, D. A. Randall, T. G. Corsetti, and D. A. Dazlich, 1989: Earth radiation budget and cloudiness simulations with a general circulation model. *J. Atmos. Sci.*, **46**, 1922-1942.

[12] Kim, Y.-J., and A. Arakawa, 1995: Improvement of orographic gravity-wave parameterization using a mesoscale gravity-wave model. *J. Atmos. Sci.*, **52**, 1875-1902.

[13] Kim, Y.-J., 1996: Representation of subgrid-scale orographic effects in a general circulation model: Part I. Impact on the dynamics of simulated January climate. *J. Climate*, **9**, 2698-2717.

[14] Elliott, S., X. Zhao, R. Turco, C-Y. Kao and M. Shen, 1995: Photochemical numerics for global scale modeling: Fidelity and GCM testing. *J. Applied Meteorology*, **34**, 694-718.

[15] Dorman, J. L., and P. J. Sellers, 1989: A global climatology of albedo, roughness length and stomatal resistance for atmospheric general circulation models as represented by the Simple Biosphere model (SiB). *J. Appl. Meteor.*, **28**, 833-855.

[16] Wehner, M. F., A. A. Mirin, P. G. Eltgroth, W. P. Dannevik, C. R. Mechoso, J. D. Farrara and J. A. Spahr, 1995: Performance of a distributed memory finite-difference atmospheric general circulation model. *Parallel Computing*, **21**, 1655-1675.

[17] Lou, J.-Z., and J. D. Farrara, 1996: Performance analysis and optimization on the UCLA parallel atmospheric model code. *Proceedings of Supercomputing '96*

[18] Chou, M. D., 1984: Broadband water vapor transmission functions for atmospheric IR flux computations. *J. Atmos. Sci.*, **41**, 1775-1778.

[19] Chou, M. D., and L. Peng, 1983: A parameterization of the absorption in the 15 mm CO_2 spectral region with application to climate sensitivity studies. *J. Atmos. Sci.*, **40**, 2183-2192.

[20] Rodgers, C. D., 1968: Some extension and applications of the new random model for molecular band transmission. *Quart. J. Roy. Meteor. Soc.*, **94**, 99-102.

[21] Randall, D. A., and V. Pan, 1993: Implementation of the Arakawa-Schubert cumulus parameterization with a prognostic cumulus kinetic energy. *The Representation of Cumulus Convection in Numerical Models of the Atmosphere,* K. A. Emanuel and D. J. Raymond, Eds., American Meteorological Society. 137-144.

[22] Drummond, L. A., J. D. Farrara, C. R. Mechoso and J. Z. Lou, 1997: Performance optimization of an atmospheric model in massively parallel computers. *Proceedings of High Performance Computing and Networking '97 (HPCN '97),* Vienna, Austria, 28-30 April 1997, 67-71.

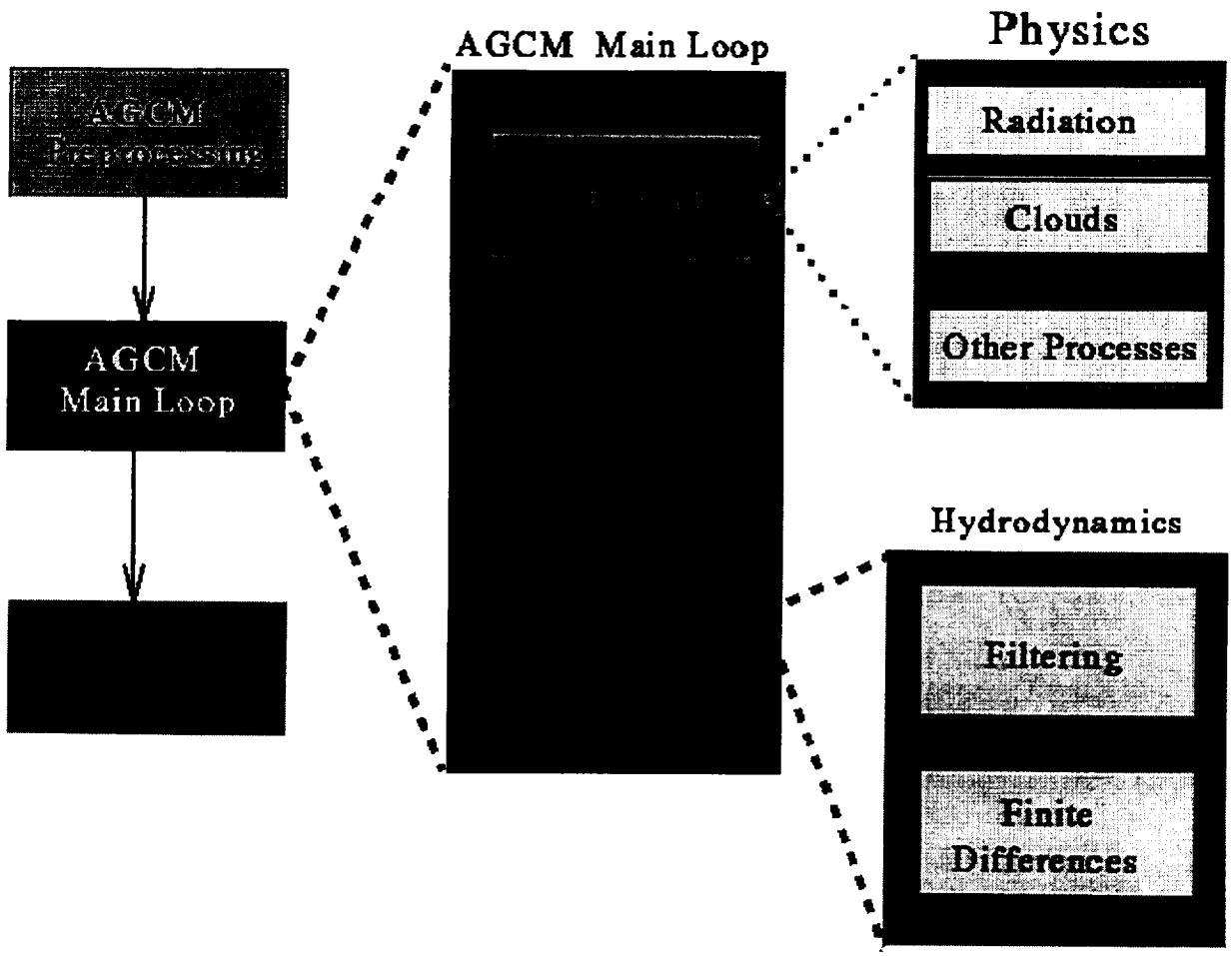


Figure 1. A schematic of the structure of the AGCM code. The main components are the Physics and Hydrodynamics (or, as in the text, Dynamics).

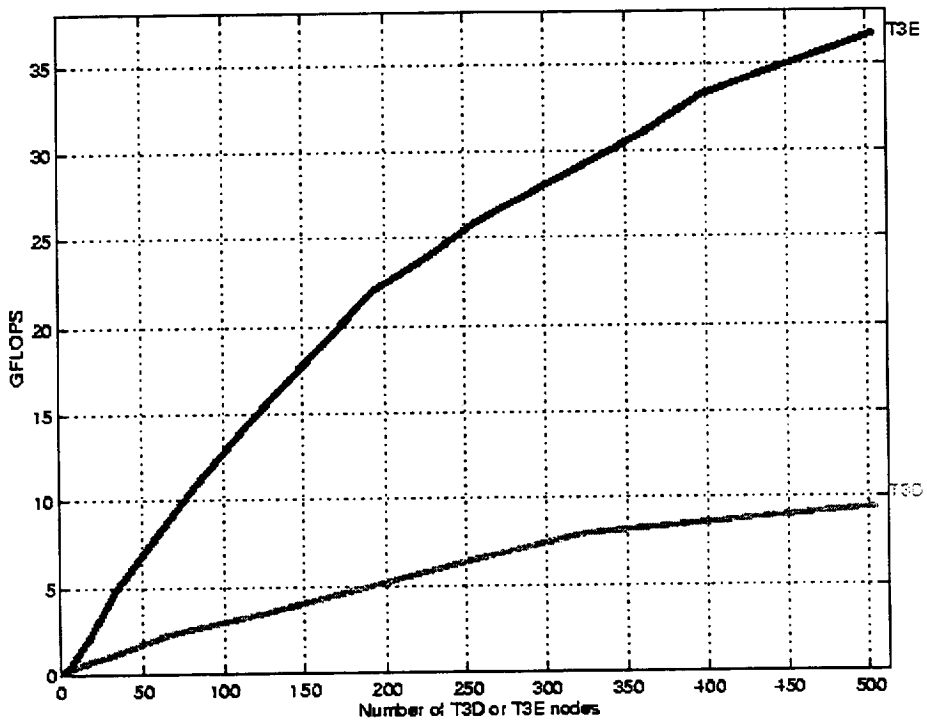


Figure 2. Performance of the AGCM/ACM on the T3D and T3E as a function of number of processors. On the T3E the model code executes approximately 8.5 times faster on 256 nodes than on 16 nodes (smallest number that will fit this size problem), resulting in a parallel efficiency of 53%. For this problem size, the parallel efficiency decreases rather rapidly above 256 nodes. The FLOPs were counted using the Apprentice toolkit on the T3D.

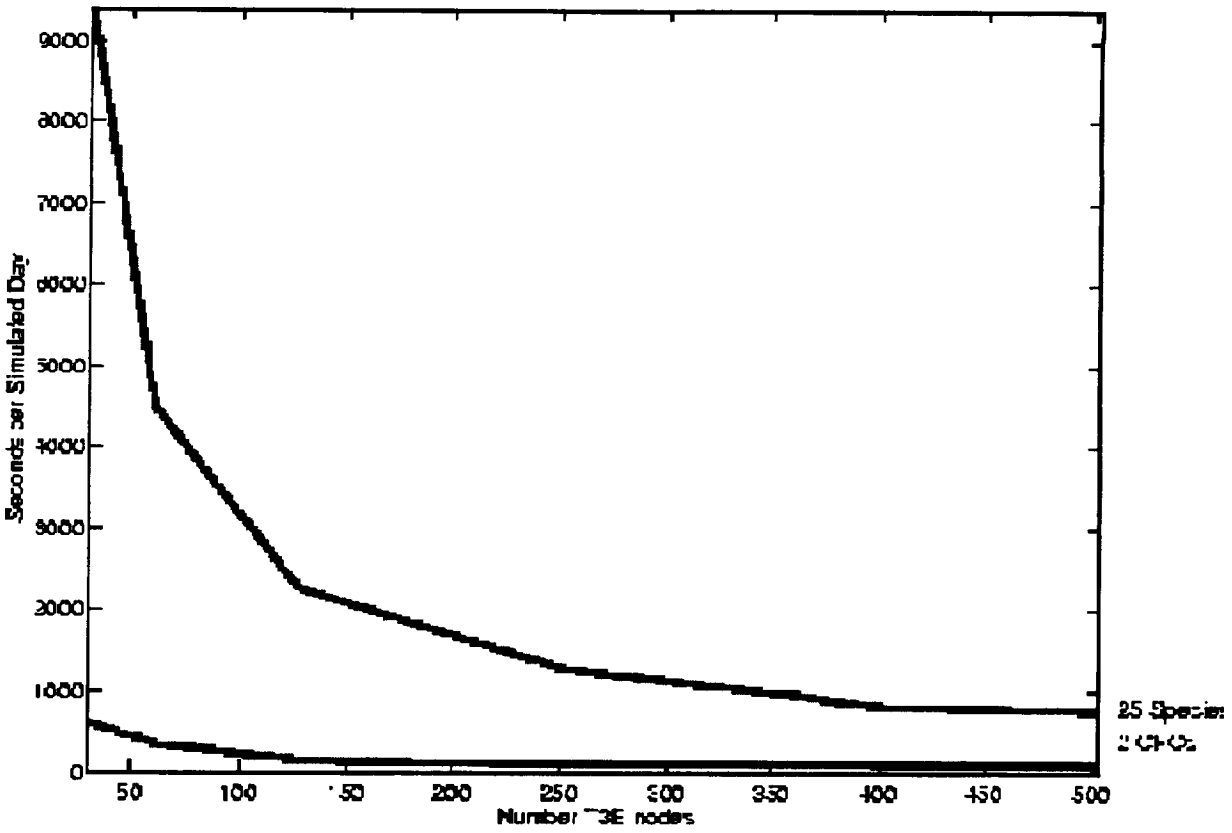


Figure 3. AGCM/ACM timings (expressed in seconds/simulated day) for two different configurations of the ACM. The lower curve corresponds to the configuration using with 2 photochemically active species (CFCs); the timings for this configuration is not very different from those for the AGCM alone. The upper curve corresponds to a configuration involving the 25 species required to simulation the evolution of stratospheric ozone. In this case, the ACM execution time becomes absolutely dominant as the total execution time is more than one order of magnitude greater in this case.

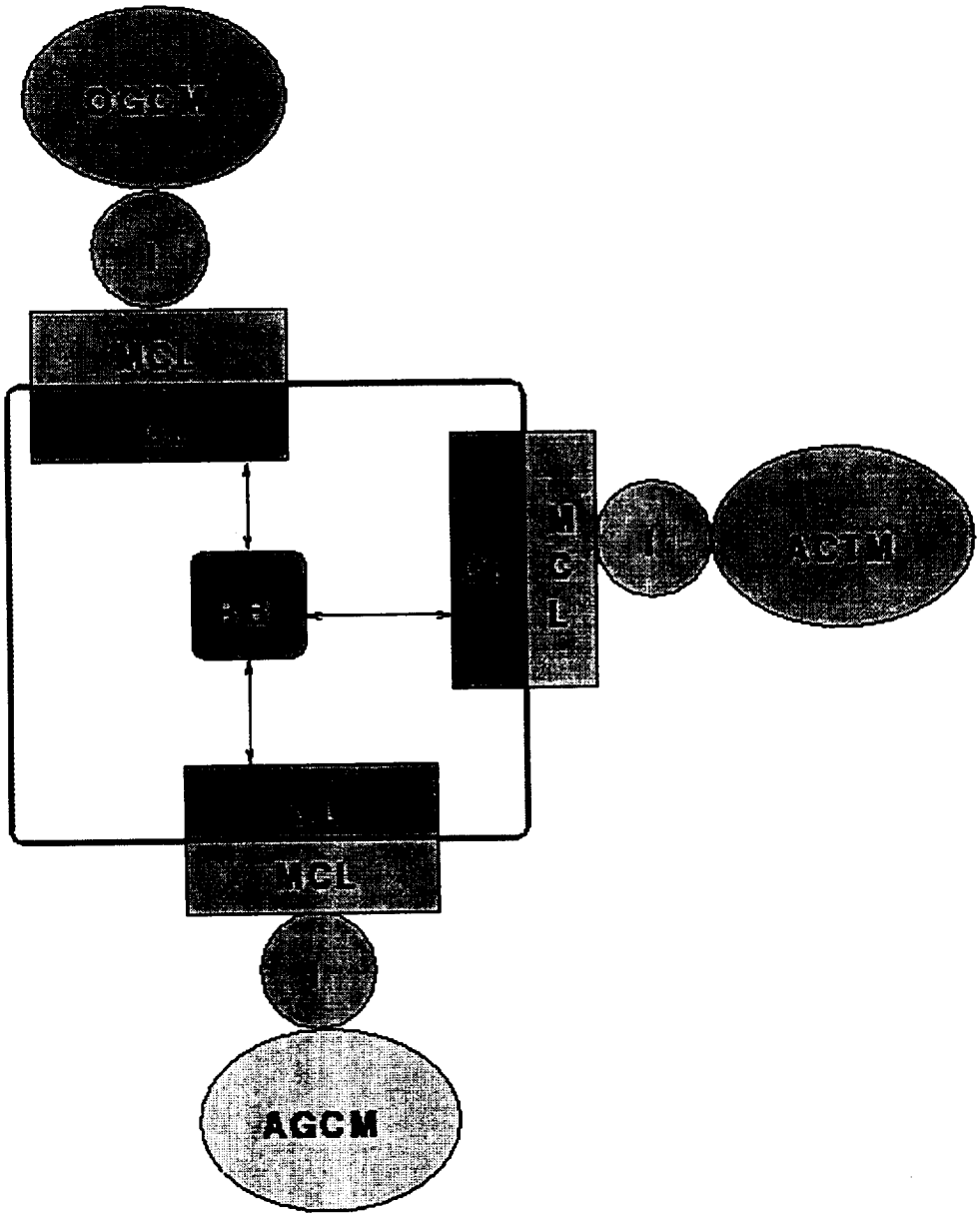


Figure 4. A schematic of the AGCM/OGCM/ACM coupled using the "Data Broker". A model communication library interface (MCL) allows the code components to exchange field variables with other components without knowing the details of how data are distributed for the other components. The communication library (CL) interface and the registration broker (RB) perform the function of moving data to/from the models. Finally, interpolation and extrapolation routines (I) are available for each model. These routines perform the various mappings and/or transformations of data from different grids or units to the desired representation.

Co-Array Fortran: current status and recent results with MICOM

Ilene Carpenter
Silicon Graphics/Cray Research
655E Lone Oak Drive
Eagan, MN 55121
ilene@cray.com
+1 612 683-3629

Robert W. Numrich
Silicon Graphics/Cray Research
655E Lone Oak Drive
Eagan, MN 55121
rwn@cray.com
+1 612 683-5481

Aaron Sawdey
Silicon Graphics/Cray Research
655A Lone Oak Drive
Eagan, MN 55121
sawdey@cray.com
+1 612 683-5872

John Reid
Rutherford Appleton Laboratory
Dept. for Computation and Information
Chilton, DIDCOT
Oxon, UNITED KINGDOM
OX11 0QX
J.K.Reid@rl.ac.uk
+44 1235 446493

Co-array Fortran [1], formerly known as F[2,3,4,5,6], is a simple extension to Fortran 90/95, which uses a second set of array subscripts to address that array in different processes. Co-array Fortran is very close in spirit to the one-sided message-passing library (SHMEM) and can be thought of as syntax for the one-sided get/put model that is incorporated into the Fortran language. It adopts a Single-Program-Multiple-Data (SPMD) programming model, in which a single program is replicated to a number of images, each with its own local data. Within each image, the normal rules of Fortran apply, as if it were a single program. Each image has a unique index associated with it and executes asynchronously. The programmer uses explicit synchronization procedures as needed.

Communication between images is done through a new type of object called a co-array. A co-array is a variable declared with dimensions in square brackets instead of, or in addition to, the dimensions declared with parentheses. Array indices in square brackets provide a convenient

notation for accessing objects across images, and follow similar rules to ordinary Fortran array indices. A co-array is declared or allocated with an asterisk for its final dimension and it always has size equal to the number of images. A reference to a co-array with no square brackets attached to it is a reference to the object in the local memory of the executing image. When communication is needed, the programmer uses the image number of the remote image in the square brackets to generate a reference to an object in the remote image. For example, the statement

```
real, dimension(n)[*]:: x,y
```

creates two real arrays of size n in each image. Because of the square brackets, these arrays are also co-arrays. The statement

```
x(5)=y(5)[q]
```

copies the value of `y(5)` in image `q` to the local value `x(5)` in the image that executes the statement. Local variables become globally visible only through co-array syntax.

Co-array syntax is more flexible than libraries or directives because it allows arbitrary Fortran 90/95 variable types and arbitrary communication patterns. For example, if different sizes are required on different images, a co-array may be declared of a derived type with a component that is a pointer array. The pointer component is allocated on each image to have the desired size for that image (or not allocated at all, if it is not needed on the image). The statement

```
x(:) = a[p]%ptr(:)
```

means 'Go to image p, obtain the pointer component of variable a, read from the corresponding pointee, and place that data in the local array x'. This flexibility may be the key to difficult problems such as adaptive mesh refinement.

Each image has its own independent I/O units. A file may be opened on one image when it is already open on another. For all units (except those identified by * in a READ or WRITE statement) each image positions each file independently. If file access order matters, the programmer needs to synchronize explicitly.

Most atmospheric and ocean models are parallelized using domain decomposition. Co-array Fortran is a natural representation of domain decomposition and provides a simple and efficient way to parallelize these codes. Consider the calculation of the maximum value of a co-array:

```
  subroutine greatest(a,great)
      ! find maximum value of a(:)[*]

      real, intent (in) :: a(:)[*]
      real, intent (out) :: great[*]
      real :: work(num_images())     ! local work array
      great = max(a(:))              ! find max of local data
      call sync_images()
      if(this_image(great) == 1) then
           work=great[:]             ! gather local max from other
images
           great[:] =max(work) ! scatter global max to all images
      end if
```

```
end subroutine greatest
```

Next, consider the task of each image getting a piece of a global array from the "master" image:

```
subroutine splat(xglobal, xlocal, lev, master)
include 'dimensions.h'
real xglobal(-nbdy+1:I_max+nbdy, -nbdy+1:j_max+nbdy)[0:1]
real xlocal(-nbdy+1:idm+nbdy, -nbdy+1:jdm+nbdy,lev)
integer lev, master

if(me.eq.master) then
     do 10 j=1-nbdy,jj+nbdy
     do 10 i=1-nbdy,ii+nbdy
       xlocal(i,j,lev)=xglobal(i+istart-1,j+jstart-1)
10   continue
else
     do 20 j=1-nbdy,jj+nbdy
     do 20 i=1-nbdy,ii+nbdy
       xlocal(I,j,lev)=xglobal(i+istart-1,j+jstart-1)[master]
20   continue
endif

return
end
```

Other examples of tasks common to this class of applications will be presented.

Portions of co-array Fortran have been incorporated into the SGI F90 compiler (versions 7.2 and 7.2.1) and the message-passing version of MICOM has been converted to use co-array Fortran. Performance of MICOM using co-array Fortran will be compared to that achieved with other programming models on SGI/Cray systems.

References:

[1] Numrich, R.W. and Reid, J. Co-array Fortran for parallel programming. Technical Report, Rutherford Appleton Laboratory, in preparation

[2] Numrich, R.W. F--: A parallel Fortran language, Technical Report, Cray Research, Inc., April 1994

[3] Numrich, R.W. F--: A parallel extension to Cray Fortran. *Scientific Programming* **6**, 275-284.

[4] Numrich, R.W. and Steidel, J.L. F--: A simple parallel extension to Fortran 90. *SIAM News*, **30**,7,1-8.

[5] Numrich, R.W. and Steidel, J.L. Simple parallel extensions to Fortran 90. *Proc. Eighth SIAM conference on parallel processing for scientific computing*, Mar. 1997

[6] Numrich, R.W., Steidel, J.L., Johnson, B.H., de Dinechin, B.D., Elsesser, G., Fischer, G., and MacDonald, T. Definition of the F-- extension to Fortran 90. Proceedings of the 10th International Workshop on Languages and Compilers for Parallel Computers, *Lectures on Computer Science Series*, Number 1366, Springer-Verlag

Metacomputing – What's in it for me?

Greg Lindahl
Andrew Grimshaw
Adam Ferrari
Katherine Holcomb

University of Virginia Computer Science Department
Thornton Hall
Charlottesville VA 22903
+1 804 982-2293
lindahl@cs.virginia.edu

Introduction

We are often asked by applications programmers, "What exactly is a metasystem, and why should I care?" This paper attempts to answer this question from an applications perspective, pointing out concrete ways in which the current practices of high performance scientific computing can possibly be improved.

What Is A Metasystem?

Before we can answer the challenge posed by the title of this paper, we need to define what a metasystem is. Physically, a metasystem is a collection of geographically separated resources (people, computers, instruments, databases) connected by a high speed network. A metasystem is distinguished from a simple collection of computers by a software layer, often called middleware, which transforms a collection of independent resources into a single, coherent, virtual machine. This machine should be as simple to use as the machine on the user's desktop, and should allow easy collaboration between colleagues located anywhere in the world.

What is the problem with today's collections of computers? A typical researcher using machines at multiple sites faces the problem of slightly or radically different software environments, separatefilesystems which require frequent copying of files between sites, security policies which prohibit transfer of files between sites without a human typing in a password, and so forth.

So why don't we have metasystems today? As usual, the fundamental difficulty is software, specifically an inadequate model of "systems software" for the worldwide collection of computer systems. Faced with the eternal rush of new hardware, the computing community has stretched existing models – interacting but autonomous computers – to a level where this model breaks down. The result is a collection of incompatible, incomplete solutions which work well in isolation, but do not work together nor scale for the future.

Our vision of a metasystem[4] is of a system containing thousands of computers and terabytes of data in a loose confederation, tied together by high-speed networks. The user will have the illusion of a very powerful computer on her desk, and will manipulate objects representing data resources such as databases of physical data, applications such as physical simulations and visualization tools, and physical devices such as scientific instruments. To allow the construction of shared workspaces, these objects may be securely shared with other users.

It is the metasystem's responsibility to support this illusion of a single machine by transparently managing data movement, caching, and conversion; detecting and managing faults; ensuring that the user's data and physical resources are adequately protected, and scheduling application components on the resources available to the user.

The potential benefits of a metasystem to the scientific community are enormous:

- more effective collaboration, by putting co-workers in the same virtual workplace,
- higher application performance due to parallel execution and exploitation of off-site resources,
- improved productivity through a considerably simpler programming environment.

The next section of this paper will introduce an example application, which we will use in the subsequent section to illustrate the benefits of the major subsystems of the Legion metacomputing system.

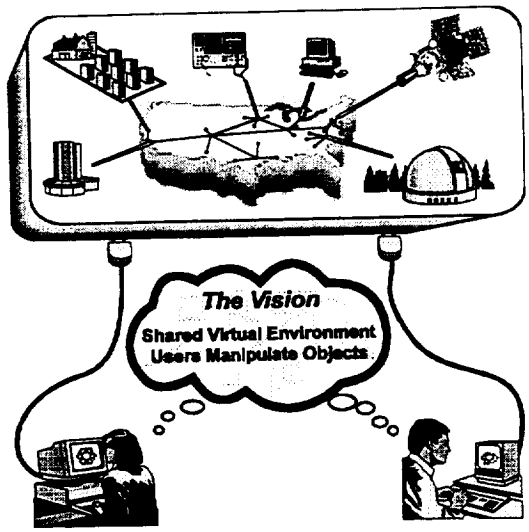


Figure 1. Our vision is to construct a shared, high-performance, secure, and fault tolerant computation environment for scientists and engineers. The resulting environment will enhance the productivity of the scientific community.

Example – Multi-Scale Climate Modeling

Climate modeling is an example of a field that can benefit greatly from metacomputing. Climate modeling has progressed beyond atmospheric simulations to include multiple aspects of the Earth system, such as full-depth ocean models, high-resolution land-surface models, sea ice models, chemistry models, and so forth. Each component model generally requires a different resolution in space and time. We might even wish to couple global and regional models. For example, an ElNino study might involve coupling a global climate code with a regional weather code.

Over the course of simulating a coupled system, the individual models must exchange data, such as temperature, winds, and precipitation, and the execution of the entire system must be kept in synchronization. The models often originate from different research groups around the world and may even be written in different languages. As an additional complication, some models have parallel implementations, often using different parallel toolkits.

With existing tools, coupling these models would be tedious and error-prone at best. Many parallel toolkits are incompatible with each other, and do not support heterogeneous collections of systems. In the metacomputing environment, the models couldinteroperate on the same or different machines, which need

not be in close physical proximity. While this solves some of the problems of coupled climate models, other difficulties remain. Fault tolerance and security are major issues. Many scientific models write restart dump files at regular intervals, and can be restarted by hand after a system failure, but this is only a partial solution. Constant restarting is an aggravation at best; at worst it is a waste of resources and researcher time, as well as an invitation to error. Security problems, which can often be neglected inside a single machine, are also potentially increased by use of far-flung resources.

A final issue is visualization. The larger and more complex the simulation, the more critical is the need for visualization, in order for humans to be able to digest the enormous amount of data generated by high-resolution scientific models. But it can be difficult to couple visualization tools to applications.

We are currently constructing such a multi-scale climate model as part of the NPACI ESS effort, linking together the UCLA OGCM/AGCM code with a regional California code.

So What Can I do With a Metasystem?

Our initial description of a metasystem is rather vague and high-level. The real question for users is: What's in it for me? There are many ways to use the new capabilities that a metasystem provides. They range from relatively simple use of Legion facilities to intricate usages which exploit Legion's capabilities to solve problems currently considered impossible. Below, we sketch out several uses of metasystem technology, and the capabilities that these features add to your research.

Shared Persistent Object (file) Space

The simplest service a metasystem provides is location-transparent access to data files, which is usually called a distributed filesystem. An ideal distributed filesystem allows a user to access a file anywhere in the world without knowing if the file is local or remote, and without involving her systems administrator. Having a shared filesystem significantly simplifies collaboration. NFS is a well-known example of a distributed filesystem[2]. However, NFS requires super-user configuration and has significant security implications, so few users are able to use NFS to access remote data, or collaborate with colleagues in remote locations. The World Wide Web provides a limited (read-only) distributed filesystem. Legion's shared object space provides shared, secure access to data files without super-user configuration.

A more powerful model than shared files is shared object spaces. Instead of just sharing files, all entities – programs, databases, instruments, etc. – can be named and shared between users. This merging of "files" and "objects" is driven by the observation that the traditional distinction between files and other objects is not necessary. Files represent persistent data, and happen to live on a disk, so files are slower than RAM, but persist if the computer crashes. In a shared object space, a file object is any object which supports the standard file operations, such as "read" and "write". In addition, the object interface can also define additional properties such as its persistence, fault, synchronization, and performance characteristics. Not all files need be the same; this eliminates the need to provide Unix synchronization semantics for all files, since many applications simply do not require those semantics. Instead, special semantics can be selected on a file-by-file basis, and even changed at run-time.

Beyond basic sequential files, persistent objects with flexible interfaces offer a range of opportunities, including:

- *Application-specific "file" interfaces* . Instead of just read and write, a "2D array file" may also have functions such as "read_column", "read_row", and "read_sub_array". The advantage to the user is the ability to interact with the file system in application terms – in this example a special object which efficiently reads and writes 2D files – rather than just one-dimensional streams of bytes. The implementations of files can be optimized for a particular data structure, by storing the data in sub-arrays, or by scattering the data to multiple devices to provide parallel I/O. Note that these characteristics can be set on a file-by-file basis, unlike most current parallel filesystems.

- *User specification of caching and prefetch strategies*. This feature allows the user to exploit application domain knowledge about access patterns and locality to tailor the caching and prefetch strategies to the application. For example, the user may know what data she might read minutes in advance.
- *Active simulations*. In addition to passive files, persistent objects may also be active entities. For example, a factory simulation can proceed at a specified pace (e.g., wall clock time) and can be accessed (read) by other objects. Of course the factory simulation may itself use and manipulate other objects.

Transparent Remote Execution

A slightly more complex service is that of transparent remote execution. Consider a user working on a code. After setting up the initial data for a run, she is left with the problem of deciding where to execute the code. She might choose to run it on her workstation (if it has sufficient resources), or on a local high performance machine, or on a remote workstation farm, or on a remote supercomputer. The choice involves many trade-offs. Which choice will result in the fastest turn-around? Today, a user must usually check each potential machine by hand to guess the turn-around time.

Next, there are the inconveniences of using remote resources. Data and executable binaries may first need to be physically copied to a remote center, and the results copied back for analysis and display. This may be further complicated by the need to access input data from a collaborator. Finally, the user must recall how to use the local queuing system; there are 25 different ones in use [3]. These inconveniences are usually so great that most users pick one site and infrequently consider moving. Finally, there are the administrative difficulties of acquiring and maintaining multiple accounts at multiple sites.

In a metasystem, the user can simply execute the application at the command line. The underlying system selects an appropriate host from among those the user is authorized to use, transfers program binaries, and begins execution. Data is transparently accessed from the shared persistent object space, and the results are stored in the shared persistent object space for later use. A queuing system could be used, in order to create a wide-area queuing system, extending today's local queuing systems.

Wide-Area Parallel Processing

Another opportunity presented by metasystems is connecting multiple resources together for the purpose of executing a single application, providing the opportunity to run problems of a much larger scale than would otherwise be possible. Not all problems will be able to exploit this capability, since the application must tolerate the latency involved in crossing a building or crossing the country.

First, let's consider a parameter space study. In a parameter space study, the same program is repeatedly executed with slightly different parameters. The program may be sequential or parallel. For example, a convergence study might involve running the same code repeatedly with different grid sizes, or with slightly perturbed initial values. These sorts of problems are sometimes called "bag of tasks" problems, since all the runs are independent.

Bag-of-tasks problems are well suited to metasystems because they are highly latency tolerant. While one computation is being performed, the results of the previous computation can be sent to the results bag, and the parameters for the next computation can be retrieved from the input bag. Furthermore, the computations can be easily spread to a large number of sites (using Legion's remote execution capability), because the computations do not interact in any way.

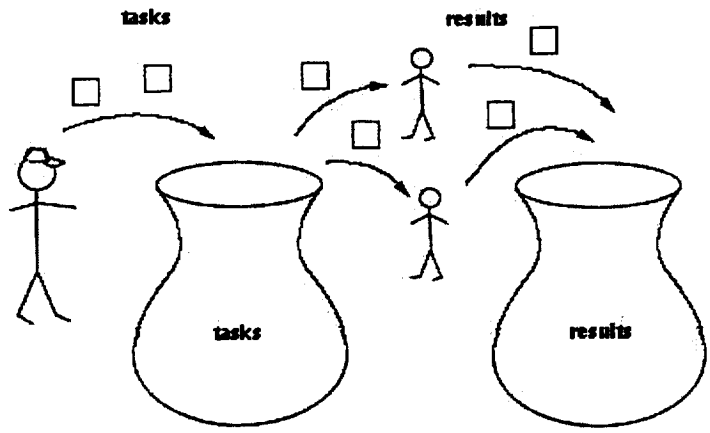


Figure 2. Bag-of-tasks. The master places work units containing parameters for the workers into the tasks bag. Workers take work from the bag, perform the required computation, place the results into another bag, perhaps used by other workers as input, and retrieve another piece of work. This continues until all of the work has been completed. Note that there may be more than one master.

Consider next a more complex class of problems, such as an extremely large ocean model[1]. Suppose that we wish to use two distributed memory MPPs and a visualization system at different sites for a single run. All 3 sites are connected by a fast network running at 155 megabits per second. (Figure 3). Further suppose that the first host has twice as many processors as the second does. Balancing the load requires that the problem be decomposed in such a manner that the first host has twice as much of the data as the second. (In general, the scheduling problem can become extraordinarily complex, so a simple example is used here to illustrate the point.)

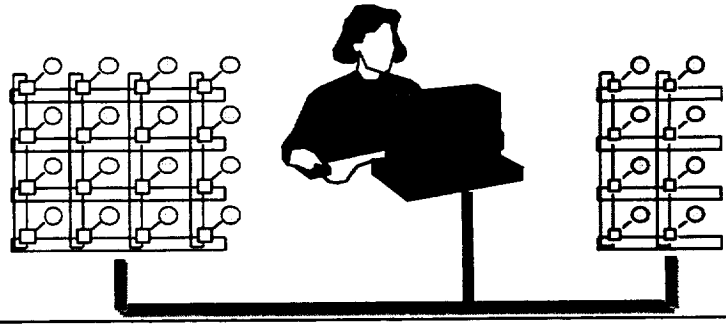


Figure 3. Two geographically separated distributed memory MPP's connected by a high-speed link. The user sits at a visualization station at a third site. The hosts may be the same or have different processors and interconnection networks

[1] Most ocean codes are 3D, but are decomposed in 2D.

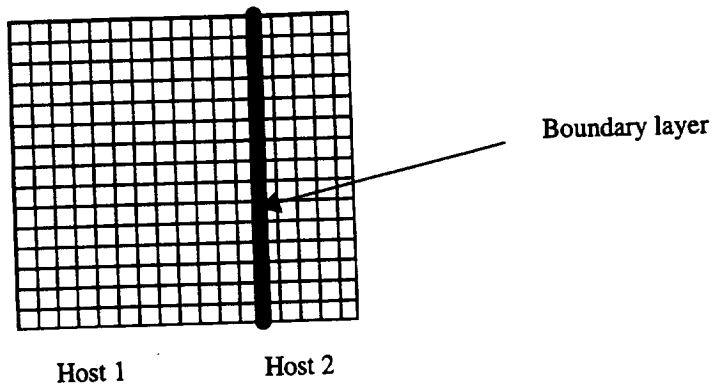
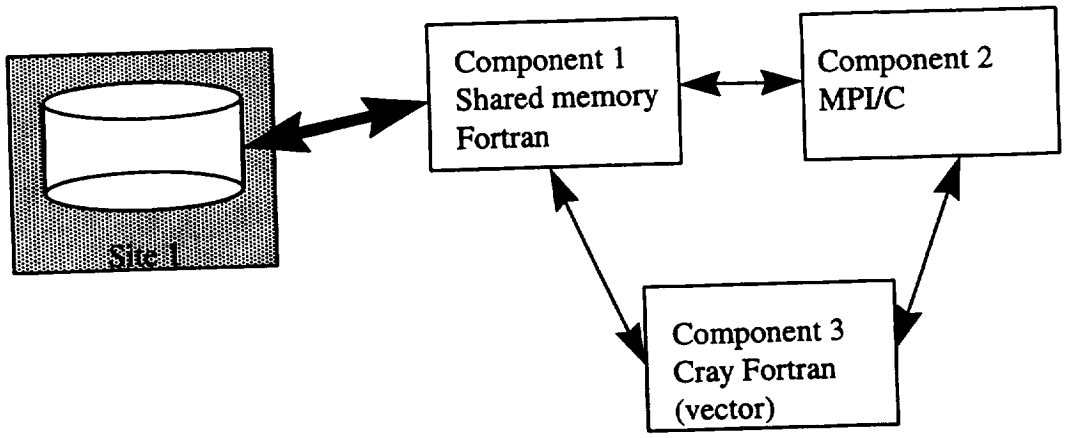


Figure 4. One possible decomposition of a 2D grid between two MPP's. The thick line represents the boundary layers that must be sent between hosts.

Given the decomposition shown above and information on the size of the zones, we can estimate the bandwidth requirements of the communications channel. Suppose that the problem is 10,000 by 10,000 zones, and each zone communicates with its 4 immediate neighbors only once per cycle, and each zone contains 100 bytes of data. Then one megabyte of data must be transferred over the wire in each direction for each cycle. Assuming coast-to-coast communication and a 155 megabit channel, the time to transmit the boundary layer is at least 50 ms. For some applications, such as those which use an implicit solver and transfer information many times during a cycle, that is likely to be too long. But for others, 50 ms is acceptable, especially if communication can overlap with computation[2].

Meta-Applications

The most challenging class of applications for the metasystem is meta-applications. A meta-application is a multi-component application, many of whose components were previously executed as stand-alone applications.



A generic example is shown above. In this example three previously stand-alone applications have been connected together to form a larger, single application. Each of the components has hardware affinities. Component 1 is a fine grain data parallel code that requires a tightly coupled machine such as anSGI Origin 2000, component 2 is a coarse grain data parallel code that can run efficiently on a "pile of PCs"

[2] Deciding the partition and placement of cells on processors can be difficult to get right if done by hand, but fortunately there are tools for making those decisions [4]. The issue of dynamically changing the partition and placement has not been solved.

machine, and component 3 is a non-parallelized but vectorizable code that "wants" to run on a vector machine such as Cray T90. Component 1 also uses a very large database that is physically located at site 1. Component 3 is written with Cray Fortran extensions, component 1 is written inHPF, and component 2 is written in C using MPI calls.

There are many difficult issues involved in this example. The first is that data is often geographically distributed – it is often stored physically close to the people who collect it, not necessarily the people who use it. Today's coupled models usually require all the data to be copied to a single location. The challenge to the metasystem is to help determine when it makes sense to move the computation to the data, and when it makes sense to move the data to the computation.

Next, scheduling the meta-application onto the hardware is a significant challenge. Consider scheduling our example meta-application on a single distributed memory machine. We would like each component to progress at the same rate, so we might need to assign different numbers of processors to each. Second, the component tasks must be mapped to the processors in such a manner as to reduce the communication load – random placement may lead to communication bottlenecks. Finally, the computational requirements of the components may vary over time, requiring dynamic re-partitioning of resources.

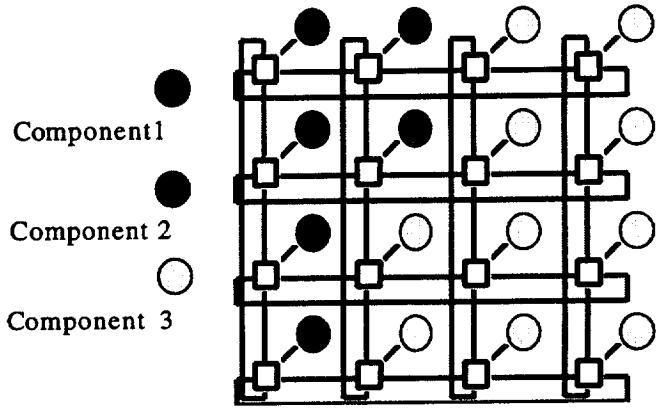


Figure 6. One possible mapping of three data-parallel components onto an MPP.

Now suppose that instead of a fixed number of processors on an MPP to choose from, we must choose between a large number of diverse systems, each connected to the others by networks of widely varying capability[3]. It is easy to see that the scheduling problem is a significant challenge.

As the number of hosts and processors in a computation increase, the mean time to failure falls. Today's large machines are less reliable than the machines used 5 years ago; collections of workstations have always presented a fault tolerance challenge. The metasystem should provide transparent fault tolerance whenever possible.

[3] Just determining network characteristics is non-trivial in a metasystem. Unlike an MPP, which is often not shared, the wide-area network is usually shared, resulting in large variances in both bandwidth and latency. Predicting network performance [5] is a critical component of the metasystem scheduling problem.

Summary

Metasystems technology is rapidly maturing. Three years ago at Supercomputing 95, the I-Way was a one-time stunt that demonstrated a large number of applications that had been constructed in an *ad hoc* fashion. Today, metasystems testbeds are operational on a full-time basis. As the technology matures further and becomes hardened enough for production use, we hope to see a significant increase in computational scientists' productivity.

Metasystems will provide users with a transparent, distributed, shared, secure, and fault-tolerant computational environment. With a metasystem, users will be able to share files, computational objects, databases, and instruments. No longer will they have to manually decide where to execute their programs and copy the necessary binaries and data files: the metasystem will do those jobs. New classes of applications, meta-applications, will be enabled by the new infrastructure, further increasing users' efficiency and productivity.

Acknowledgments: This work partially supported by DARPA(Navy) contract # N66001-96-C-8527, DOE grant DE-FD02-96ER25290, DOE contract Sandia LD-9391, Northrup-Grumman (for the DoD HPCMOD/PET program), DOE D459000-16-3C and DARPA (GA) SC H607305A. The authors would also like to thank Sarah Parsons Wells for her editing assistance.

References

1. A. Grimshaw and W. Wulf, "The Legion Vision of a Worldwide Virtual Computer," *Communications of the ACM*, pp. 39-45, vol. 40, number 1, January, 1997.
2. E. Levy, and A. Silberschatz, 'Distributed File Systems: Concepts and Examples," ACM Computing Surveys, vol. 22, No. 4, pp. 321-374, December, 1990.
3. J.A. Kaplan and M.L. Nelson, "A Comparison of Queueing, Cluster, and Distributed Computing Systems," NASA Technical Memorandum 109025, NASA LaRC, October, 1993.
4. J. B. Weissman, A.S. Grimshaw, "A Framework for Partitioning Parallel Computations in Heterogeneous Environments ", Concurrency: Practice and Experience, pp. 455-478, Vol. 7(5), August, 1995.
5. R. Wolski, 'Dynamically Forecasting Network Performance to Support Dynamic Scheduling Using the Network Weather Service," *6th IEEE Symposium on High Performance Distributed Computing*, 1996.

Impact of Commmunication Protocol on Performance

Patrick H. Worley

Oak Ridge National Laboratory
P.O. Box 2008
Oak Ridge, Tennessee 37831-6367
email: worleyph@ornl.gov
phone: +1 423 574-3128
Fax: +1 423 574-0680

1. Introduction. On previous generation MPP systems, interprocessor communication often represented a significant fraction of the runtime of production parallel codes, and the choice of communication transport layer and communication protocol were important steps in porting and tuning application codes. Processor, network, and transport layer performance continue to improve, and the sensitivity of performance to these implementation issues needs to be reexamined.

In this paper we use the PSTSWM parallel application code to examine
1) single processor performance,
2) peak achievable point-to-point communication performance,
3) performance variation of kernels as a function of communication protocols,
4) performance of vendor-supplied collective communication routines, and
5) performance sensitivity of full code to choice of parallel implementation

for the SGI/Cray Research T3E and Origin 2000, using both the MPI [2] and SHMEM libraries to implement interprocessor communication. While other researchers have looked at communication performance on these machines (e.g., [1]), this study differs in that we examine the effect on peformance of an application code.

2. PSTSWM. The Parallel Spectral Transform Shallow Water Model (PSTSWM) is a message-passing parallel benchmark code and parallel algorithm testbed that solves the nonlinear shallow water equations on a rotating sphere using the spectral transform method. PSTSWM was developed by the author and by I. T. Foster at Argonne National Laboratory from the serial code STSWM, written by J. J. Hack and R. Jakob at the National Center for Atmospheric Research. PSTSWM was used to evaluate parallel algorithms for the spectral transform method as it is used in global atmospheric circulation models.

PSTSWM has characteristics that make it useful for performance studies. It makes interesting and varied demands on the communication subsystem, multiple parallel algorithms are embedded in the code, and multiple message-passing transport layers are supported. See `http://www.epm.ornl.gov/chammp/pstswm/index.html` for a

partial bibliography of the performance studies utilizing PSTSWM.

3. Platforms. We focus on the T3E and Origin2000 in these studies, but include measurements from the following platforms to aid in the understanding of the results.

Intel Paragon XP/S 150 at Oak Ridge National Laboratory.
This machine has 1024 MP nodes (3 50-MHz iPSC/860 processors per node). Measurements were taken in January, 1998. Only one processor per node was used for computation. KAI math routines were used.

CRI T3D at Cray Research in Eagen, MN.
This machine had 128 150-MHz DEC Alpha EV4 processors. Measurements were taken in August, 1996.

IBM SP2 at NASA Ames Research Center.
This machine had 160 RS6000/590 nodes ("wide", 66.7 MHz POWER2). Measurements were taken in August, 1996. ESSL math routines were used.

Convex SPP-1200 at the National Center for Supercomputer Applications.
This machine has 64 120-MHz HP PA-RISC 7200 processors (8 Hypernodes). Measurements were taken in September, 1996.

SGI/CR T3E-900 at the National Energy Research Scientific Computing Center.
This machine has 532 450-MHz DEC Alpha EV5 RISC processors. Measurements were taken in January, 1998.

HP/CONVEX SPP-2000 at the National Center for Supercomputer Applications.
This machine has 64 180-MHz HP PA-RISC 8000 processors (4 Hypernodes). Measurements were taken in April, 1998. VECLIB math routines were used.

Intel PII-266 cluster at Oak Ridge National Laboratory.
This machine has 10 266-MHz dual Pentium II nodes. Measurements were taken in February, 1998. LINUX and Portland Group f77 compiler were used.

SGI/CR Origin2000 at Los Alamos National Laboratory.
This machine has 128 195-MHz MIPS R10000 processors. Measurements were last taken in April, 1998. SCSL math routines were used.

4. Serial Performance. Table 1 contains the MFlop/sec rates for one processor runs of the code PSTSWM for a number of different problem sizes. PSTSWM computes the solution by timestepping, advancing the approximation to a new timelevel (in simulation time) by using the approximations at the two previous timelevels. The computational complexity and code executed for a timestep in PSTSWM are identical for all timesteps.

We use the standard benchmark problem for the shallow water equations, global steady state nonlinear zonal geostrophic flow [3], and two problem size classes: T42

and T85, characterized by the following computational grids and complexity.

	physical grid	Fourier grid	spectral coefficients	flops per timestep
T42	64 × 128	64 × 64	946	4129859
T85	128 × 256	128 × 128	3741	24235477

There is also a vertical component to the problem size. For example, T42L16 is a T42 horizontal grid with 16 vertical levels. The complexity of solving the problem is linear in the number of vertical levels.

64-bit precision floating point computation is used in all experiments. Math library routines are used for the Fourier transforms where available, as indicated in the description of the platforms in the previous section.

	T42L1	T42L3	T42L16	T85L1	T85L3
Intel Paragon	13.9	14.0	13.9	13.1	13.1
CRI T3D	25.2	25.7	23.3	24.9	24.4
IBM SP2	98.3	98.3	91.0	107.7	102.4
Convex SPP-1200	24.9	23.2	22.9	24.2	24.0
SGI/CR T3E-900	79.4	70.0	64.1	84.7	70.7
HP/Convex SPP-2000	138.8	107.3	83.5	117.5	114.2
Intel PII-266 cluster	45.4	37.2	30.3	38.9	33.7
SGI/CR Origin2000	153.0	140.8	92.5	131.7	130.1

TABLE 1
Serial MFlop/sec rates.

From this data it is clear that the serial performance of MPP processors has generally improved over the past few years, and that optimized math libraries are important performance enhancers. Also note that some effects of the memory hierarchy on performance can be observed from the variation in MFlop/sec rate as the problem size varies.

5. Point-to-Point Communication Performance. Communication overhead is best measured in the context of the full code, but it is useful to establish a performance baseline by determining the "peak achieveable" point-to-point interprocessor communication performance. Performance-critical interprocessor communication in PSTSWM is implemented using two basic types of commands: SWAP and SENDRECV. The message-passing transport layer used to implement these commands is specified at compile time, while the protocol used is specified at runtime.

To characterize the basic communication capabilities in terms relevant to PSTSWM, we use the PSTSWM SWAP command. We measure the time required to exchange 262144 64-bit floating point numbers between two neighboring processors, varying the protocol used for the exchange to find the minimum. We refer to these as the 2MB experiments. We also measure the time to swap 1024 and 16384 64-bit values, referring to these as the 8KB and 128KB experiments, respectively.

Two general classes of protocols are used: unordered (ping-ping) and ordered (ping-pong). While not all protocols are available for all message-passing transport layers, they are drawn from those described in Table 2. Examples are given using MPI commands. Note that the examples have been simplified (to save room) and do not accurately represent the MPI implementations.

Table 3 contains the maximum observed bandwidth and typical SWAP overhead ("latency") for the corresponding communication protocol. (Note that this protocol does not necessarily have the smallest latency.) The following observations on communication performance on the T3E and the Origin2000 can be drawn from this summary data:

- The T3E and the Origin2000 demonstrate significant performance improvement over previous generation MPPs of like architecture. (Note however that the SPP-2000 performance is better than both, for these particular tests.)
- SHMEM achieves considerably higher bandwidth and lower latency than MPI, but MPI performance is still an improvement over what was achieveable on earlier systems.

Looking at the raw timing data, we can also determine the sensitivity of performance to the choice of communication protocol. On the T3E the achievable bandwidth shows little sensitivity to the communication protocol when using MPI, and the simple protocols are generally slightly better. On the Origin2000, MPI performance is somewhat more sensitive to the communication protocol, but the communication protocol is still not too important. This is a significant difference from earlier results on the Intel Paragon and the IBM SP2, but is similar to the T3D results, and appears to reflect the SGI/CR implementation of MPI. When using SHMEM, the variability is higher (for both systems).

6. Parallel Algorithm Performance. Some indication of the impact of communication protocol on performance can be seen from the point-to-point communication tests, but it is difficult to use these results to predict the effect on application code performance. Here we examine this issue in more detail, looking at the effect on the performance of specific parallel algorithm options in PSTSWM.

Unordered		Ordered	
(0,0): simple		(1,0): simple	
Processors 1 and 2		Processor 1	Processor 2
MPI_BSEND		MPI_SEND	MPI_RECV
MPI_RECV		MPI_RECV	MPI_SEND
(0,1): nonblocking send		(1,1): nonblocking send	
Processors 1 and 2		Processor 1	Processor 2
MPI_ISEND		MPI_ISEND	MPI_RECV
MPI_RECV		MPI_RECV	MPI_SEND
(0,2): nonblocking receive		(1,2): nonblocking receive	
Processors 1 and 2		Processor 1	Processor 2
MPI_IRECV		MPI_IRECV	MPI_RECV
MPI_SEND		MPI_SEND	MPI_SEND
(0,3): nonblocking send & receive		(1,3): nonblocking send & receive	
Processors 1 and 2		Processor 1	Processor 2
MPI_IRECV		MPI_IRECV	MPI_RECV
MPI_ISEND		MPI_ISEND	MPI_SEND
(0,4): ready send		(1,4): ready send	
Processors 1 and 2		Processor 1	Processor 2
MPI_IRECV		MPI_IRECV	MPI_RECV
MPI_RSEND		MPI_RSEND	MPI_RSEND
(0,5): nonblocking ready send		(1,5): nonblocking ready send	
Processors 1 and 2		Processor 1	Processor 2
MPI_IRECV		MPI_IRECV	MPI_RECV
MPI_IRSEND		MPI_IRSEND	MPI_RSEND
(0,6): native sendrecv		(1,4): synchronous	
Processors 1 and 2		Processor 1	Processor 2
MPI_SENDRECV		—	MPI_RECV
		MPI_SEND	—
		—	MPI_SEND
		MPI_RECV	—

TABLE 2
SWAP protocols (simplified).

	Unordered								
	2MB			128KB			8KB		
	BW	lat.	prot.	BW	lat.	prot.	BW	lat.	prot.
Paragon									
: MPI	73	82	(0,6)	70	136	(0,3)	48	139	(0,3)
: NX	76	32	(0,3)	71	83	(0,3)	57	74	(0,3)
: SUNMOS	293	63	(0,3)	—	—	—	—	—	—
T3D									
: SHMEM	183	19	(0,2)	—	—	—	—	—	—
SP2									
: MPI	96	136	(0,4)	—	—	—	—	—	—
SPP-1200									
: MPI	45	104	(0,4)	—	—	—	—	—	—
T3E-900									
: MPI	286	30	(0,2)	245	24	(0,0)	66	25	(0,2)
: SHMEM	543	7	(0,1)	494	7	(0,1)	258	7	(0,1)
SPP-2000									
: MPI	654	39	(0,6)	629	15	(0,6)	145	23	(0,2)
Origin2000									
: MPI	142	39	(0,1)	128	29	(0,6)	57	30	(0,1)
: SHMEM	287	15	(0,1)	222	14	(0,1)	114	12	(0,2)

	Ordered								
	2MB			128KB			8KB		
	BW	lat.	prot.	BW	lat.	prot.	BW	lat.	prot.
Paragon									
: MPI	118	75	(1,0)	107	105	(1,3)	52	82	(1,0)
: NX	118	50	(1,0)	114	62	(1,0)	75	52	(1,0)
: SUNMOS	154	35	(1,0)	—	—	—	—	—	—
T3D									
: SHMEM	126	12	(1,2)	—	—	—	—	—	—
SP2									
: MPI	71	74	(1,1)	—	—	—	—	—	—
SPP-1200									
: MPI	29	27	(1,3)	—	—	—	—	—	—
T3E-900									
: MPI	163	29	(1,6)	134	20	(1,0)	47	21	(1,0)
: SHMEM	336	9	(1,2)	340	5	(1,2)	210	5	(1,2)
SPP-2000									
: MPI	541	20	(1,0)	547	9	(1,0)	158	5	(1,0)
Origin2000									
: MPI	126	33	(1,5)	98	17	(1,3)	40	17	(1,0)
: SHMEM	166	8	(1,1)	140	8	(1,1)	71	10	(1,2)

TABLE 3

Peak observed bandwidth (MBytes/sec) and latency (microseconds) for optimal protocol.

In the spectral transform method used in PSTSWM, fields are transformed at each timestep between the physical (longitude-latitude-vertical) domain and the Fourier (wavenumber-latitude-vertical) domain using Fourier transforms in the longitude direction, and between the Fourier and spectral (spectral coefficients - vertical) domains using a Legendre transform in the latitude direction. All parallel algorithms in PSTSWM are based on decomposing the different computational domains onto a logical two-dimensional grid of processors, PX×PY. In each domain, two of the domain dimensions are decomposed across the processor grid, for example, assigning longitude-latitude patches of the physical domain to individual processors, but leaving one domain dimension undecomposed.

Two general types of parallel algorithms are used in PSTSWM: transpose and distributed. In a transpose algorithm, the decomposition is "rotated" before a transform begins, to ensure that all data needed to compute a particular transform is local to a single processor. In a distributed algorithm the original decomposition of the domain is retained, and communication is performed to allow the processors to cooperate in the calculation of a transform.

Three transpose algorithms are examined, each of which is functionally equivalent to MPI_ALLTOALLV:

- **srtrans**: sends P-1 messages using SENDRECV to transpose across P processors;
- **swtrans**: sends P-1 messages using SWAP to transpose across P processors;
- **logtrans**: sends $\Theta(\log P)$ messages using SWAP to transpose across P processors.

Each of these are options for both the parallel Fourier and parallel Legendre transform algorithms. Here we restrict our study to transpose-based parallel Fast Fourier transform algorithms. One distributed Fast Fourier transform is also examined:

- **dfft**: sends $\Theta(\log P)$ messages using SWAP to calculate Fourier transform distributed across P processors.

The distributed Legendre transform algorithms in PSTSWM are based on the evaluation of distributed vector sums. Four distributed vector sum algorithms are examined, the first three of which are functionally equivalent to MPI_ALLREDUCE:

- **exchsum**: an exchange-based algorithm implemented using SWAP;
- **halfsum**: a recursive halving-based algorithm implemented using SWAP;
- **ringsum**: a ring-based algorithm implemented using SENDRECV;
- **ringpipe**: a pipeline-based algorithm implemented using SENDRECV.

Each of these algorithms can be implemented using the protocols described in Table 2. Two different types of implementations are also supported. The first uses the basic SWAP and SENDRECV commands to exchange the data. The second reorders

the elements of the SWAP or SENDRECV protocol in an attempt to overlap communication with computation and to hide communication latency. These algorithms and protocols are described in more detail in [5]. The overlap algorithms using unordered and ordered communication protocols will be designated by $(2, x)$ and $(3, x)$ respectively, where $x \in \{1, 2, 3, 4, 5, 6\}$.

To examine the performance issues in these different implementation options, we run the following experiments. We use one-dimensional decompositions of the form 8x1 or 1x8 and 32x1 or 1x32, where the first decomposition in each pair is for examining parallel Fourier transform algorithms, and the second is for examining parallel Lengendre transform algorithms. The problem sizes are based on T42L16 and T85L32 as they would appear on a two-dimensional processor grid of size 8x8, 16x32, or 32x16. This is accomplished by modifying the problem size to achieve the desired granularity (problem size per processor), and allows us to examine the performance for problem granularities that are typical of what would be seen in practice.

Results are presented in Table 4. The first column is the overall best protocol for each parallel algorithm. Multiple protocols are given when no single protocol is good for all problem sizes and numbers of processors. The other columns indicate how much performance is lost by using the MPI_SENDRECV-based protocol instead of the optimal MPI protocol and by using the optimal MPI protocol instead of the optimal SHMEM protocol. Note that these are total runtimes, and that the indicated performance loss is a function of both the size of the messages and the communication/computation ratio for a given experiment.

From this data it is clear that there is no reason to use anything but (0,6) on the T3E if using MPI, but that significant performance gains are possible if the SHMEM library is used instead. Note that, unlike with MPI, the overlap algorithms are optimal for some of the SHMEM experiments, indicating that overlap logic can be useful with this architecture if the message-passing library supports it.

On the Origin2000, the conclusions are less clear. While (0,6) is rarely optimal, it is a good choice for all but a few cases. For those few cases, however, it should not be used. Similarly, MPI is competitive with (or better than) SHMEM in most cases, but MPI performs much worse than SHMEM for some of the smaller granularity cases.

7. Full Simulation Performance. Efficient parallelizations of PSTSWM exploit two-dimensional decompositions of the domain, parallelizing both the Fourier and Legendre transforms. Here we consider two classes of parallel algorithms.

- **DTH**: double transpose for the Fourier transform and **halfsum** for the Legendre

T3E

	opt. protocols	$\frac{t_{(0,6),\text{mpi}} - t_{\text{opt,mpi}}}{t_{\text{opt,mpi}}}$				$\frac{t_{\text{opt,mpi}} - t_{\text{opt,shmem}}}{t_{\text{opt,shmem}}}$			
		$P = 8$		$P = 32$		$P = 8$		$P = 32$	
		T42	T85	T42	T85	T42	T85	T42	T85
dfft	(0,6),(2,2)	0%	8%	0%	0%	18%	6%	54%	18%
exchsum	(0,6)	0%	0%	0%	0%	7%	3%	30%	16%
halfsum	(0,6)	0%	0%	0%	0%	7%	2%	39%	11%
logtrans	(0,6)	0%	0%	1%	0%	16%	7%	82%	20%
ringpipe	(0,6)	0%	0%	0%	0%	14%	4%	75%	39%
ringsum	(0,6)	0%	0%	0%	0%	11%	3%	87%	24%
srtrans	(0,6)	2%	0%	0%	0%	17%	5%	121%	27%
swtrans	(0,6)	1%	0%	0%	0%	17%	5%	143%	29%

Origin2000

	opt. protocols	$\frac{t_{(0,6),\text{mpi}} - t_{\text{opt,mpi}}}{t_{\text{opt,mpi}}}$				$\frac{t_{\text{opt,mpi}} - t_{\text{opt,shmem}}}{t_{\text{opt,shmem}}}$			
		$P = 8$		$P = 32$		$P = 8$		$P = 32$	
		T42	T85	T42	T85	T42	T85	T42	T85
dfft	(0,1),(3,3)	1%	22%	0%	10%	7%	7%	-4%	-10%
exchsum	(0,4),(0,6)	0%	20%	0%	48%	-4%	0%	-11%	32%
halfsum	(0,1),(0,5)	1%	0%	5%	27%	3%	2%	5%	-5%
logtrans	(0,6),(1,1)	0%	9%	0%	3%	6%	14%	59%	23%
ringpipe	(2,1),(2,2)	6%	4%	1%	3%	0%	-5%	78%	3%
ringsum	(0,1)	2%	1%	1%	4%	0%	-9%	93%	-7%
srtrans	(0,1)	0%	0%	0%	1%	-1%	-3%	115%	35%
swtrans	(0,1)	1%	0%	0%	1%	0%	-3%	116%	37%

TABLE 4

Effect of protocol on performance of parallel algorithms.

transform. The double transpose algorithm uses a transpose to serialize the Fourier transforms, then another transpose to return to a domain decomposition analogous to the original. This approach has the best load balance among the parallel algorithm options. **halfsum** is the best MPI_ALLREDUCE-equivalent algorithm on the T3E and the Origin2000.

- **DR: dfft/ringpipe**. This parallel aglorithm combination has good load balance, requires the minimum storage, and has the maximum potential for communication/computation overlap.

DTH and **DR** stress the underlying transport mechanisms in significantly different ways, and represent different tests of the communication protocol sensitivity. Due to their good load balances, the performance differences between them reflect primarily the differences in communication costs.

For each platform we measure the runtimes when solving T42L16 and T85L16 using

- *opt*: the best transpose algorithms (for **DTH**) and the best communication protocols for each parallel algorithm, determined empirically,
- *gen*: **srtrans** (for **DTH**) and (0,6)-based parallel implementations, and
- *coll*: MPI collective communication routines MPI_ALLTOALLV and MPI_REDUCEALL (for **DTH**),

for logical processor meshes of sizes: 4×4, 4×8, 8×8, 8×16, 16×16, and 16×32. Algorithms *gen* and *coll* represent the typical algorithm choices if nothing is known about the communication protocol sensitivities. Measurements are also taken using 8×14 for **DR** and 14×8 for **DTH**, since the 128 processor experiments do not run efficiently on a 128 processor Origin2000 (due to competition with system processes).

Results are presented in Table 5. The optimal times are given for both MPI and SHMEM implementations. Additionally, the performance degradation (if any) is given for using the *gen* and *coll* implementations instead of the optimal MPI implementation.

- *T3E results.* For **DTH**, *gen* is the best MPI implementation except for the smallest granularity cases. In those two cases *coll* is the best, but *coll* is an erratic performer in general. For **DR**, *gen* is never the best, and it is worthwhile searching for the optimal MPI protocol. But the optimal SHMEM implementations are faster than the optimal MPI implementations in all cases, and often significantly so.
- *Origin2000 results.* For **DTH**, *gen* is a reasonable choice for the T42L16 cases, but the optimal MPI protocols are worth identifying for the T85L16 cases. *coll* is never a good choice. For **DR**, *gen* is a poor choice, and it is worthwhile searching for the optimal MPI protocols. The optimal SHMEM implementations are faster than the optimal MPI implementations only for the largest granularity cases. In the other cases, the optimal MPI implementations are consistently better.

8. Summary. Both the T3E and the Origin2000 results indicate the importance of considering the interprocessor communication protocols when tuning performance, but the similarity in the results ends there. On the T3E, performance is optimized by using the SHMEM communication library. On the Origin2000, optimization should include both the communication library (MPI or SHMEM) and the particular protocol used in the implementation. Disappointingly, the collective communication-based implementation *coll* is not competitive on either platform, which is consistent with earlier evaluations on other parallel platforms [4].

T3E

alg.	prot.	library	4 × 4	4 × 8	8 × 8	8 × 14	8 × 16	16 × 16	16 × 32
			T42L16 runtimes						
DTH	opt	MPI	30.9	14.8	7.4	4.5	4.1	2.5	2.0
	opt	SHMEM	29.6	13.9	6.5	3.5	3.2	2.0	1.2
DR	opt	MPI	23.6	12.5	7.3	6.9	5.7	4.4	-
	opt	SHMEM	22.2	11.2	6.0	5.2	4.0	2.8	-
			T85L16 runtimes						
DTH	opt	MPI	311.3	149.8	69.8	38.6	36.2	20.6	12.4
	opt	SHMEM	304.6	144.3	65.6	36.4	33.4	17.0	9.4
DR	opt	MPI	228.9	116.3	61.7	39.1	37.4	21.5	18.9
	opt	SHMEM	221.1	110.8	57.0	35.7	29.5	16.9	12.6
			T42L16 MPI performance sensitivity						
DTH	gen	MPI	0%	0%	3%	0%	0%	17%	13%
	coll	MPI	1%	5%	0%	7%	7%	0%	0%
DR	gen	MPI	5%	8%	3%	2%	4%	2%	-
			T85L16 MPI performance sensitivity						
DTH	gen	MPI	0%	0%	0%	0%	0%	0%	0%
	coll	MPI	1%	6%	5%	17%	16%	2%	23%
DR	gen	MPI	12%	7%	4%	4%	7%	8%	7%

Origin2000

alg.	prot.	library	4 × 4	4 × 8	8 × 8	8 × 14	8 × 16	16 × 16	16 × 32
			T42L16 runtimes						
DTH	opt	MPI	17.5	9.9	5.9	4.9	-	-	-
	opt	SHMEM	18.7	10.4	6.0	-	-	-	-
DR	opt	MPI	18.3	10.3	6.8	6.2	-	-	-
	opt	SHMEM	18.6	10.6	7.2	7.6	-	-	-
			T85L16 runtimes						
DTH	opt	MPI	250.9	126.4	52.5	42.6	-	-	-
	opt	SHMEM	244.1	112.7	54.8	-	-	-	-
DR	opt	MPI	250.6	122.5	56.2	40.1	-	-	-
	opt	SHMEM	234.0	104.8	56.4	54.5	-	-	-
			T42L16 MPI performance sensitivity						
DTH	gen	MPI	0%	1%	3%	4%	-	-	-
	coll	MPI	28%	37%	14%	16%	-	-	-
DR	gen	MPI	36%	18%	3%	5%	-	-	-
			T85L16 MPI performance sensitivity						
DTH	gen	MPI	2%	5%	13%	17%	-	-	-
	coll	MPI	11%	30%	120%	111%	-	-	-
DR	gen	MPI	14%	35%	92%	24%	-	-	-

TABLE 5

Runtimes of 5 day simulations of PSTSWM (seconds) and performance degradation from using gen and coll algorithms: $(t_{\text{gen,mpi}} - t_{\text{opt,mpi}})/t_{\text{opt,mpi}}$ *and* $(t_{\text{coll,mpi}} - t_{\text{opt,mpi}})/t_{\text{opt,mpi}}$.

9. Acknowledgements. This research was supported by the U.S. Department of Energy under Contract DE-AC05-96OR22464 with Lockheed Martin Energy Research Inc. We thank NASA-Ames for access to their SP2 system, and Cray Research for access to a T3D system. We thank the Advanced Computing Laboratory at Los Alamos National Laboratory for access to the SGI/Cray Research Origin2000. The Intel XP/S 150 MP Paragon operated by the Center for Computational Science at ORNL is funded by the Department of Energy's Mathematical, Information and Computational Sciences Division of the Office of Computational and Technology Research. Access to the CONVEX Exemplar SPP-1200 and the HP/CONVEX Exemplar SPP-2000 was supported by the National Center for Supercomputing Applications, University of Illinous at Urbana-Champaign under grant number ASC960028N. Access to the SGI/Cray Research T3E-900 at the National Energy Research Scientific Computing Center was supported by the Environmental Sciences Division, U.S. Department of Energy.

REFERENCES

[1] G. R. LUECKE, J. J. COYLE, AND W. UL HAQUE, *Comparing communication performance of MPI on the Cray Research T3E-600 and IBM SP-2*, Performance Evaluation and Modelling of Computer Systems, (1997). http://hpc-journals.ecs.soton.ac.uk/PEMCS/.

[2] MPI COMMITTEE, *MPI: a message-passing interface standard*, Internat. J. Supercomputer Applications, 8 (1994), pp. 165–416.

[3] D. L. WILLIAMSON, J. B. DRAKE, J. J. HACK, R. JAKOB, AND P. N. SWARZTRAUBER, *A standard test set for numerical approximations to the shallow water equations on the sphere*, Tech. Report ORNL/TM–11773, Oak Ridge National Laboratory, Oak Ridge, TN, 1991.

[4] P. H. WORLEY, *MPI performance evaluation and characterization using a compact application benchmark code*, in Proccedings of the Second MPI Developers Conference and Users' Meeting, IEEE Computer Society Press, Los Alamitos, CA, 1996, pp. 170–177.

[5] P. H. WORLEY AND B. TOONEN, *A users' guide to PSTSWM*, Tech. Report ORNL/TM–12779, Oak Ridge National Laboratory, Oak Ridge, TN, July 1995.

Parallelization Agent: A Knowledge-based System

Suraj C. Kothari
Computer Science Department
Iowa State University, Ames, Iowa 50011
Email:kothari@cs.iastate.edu
Phone: +1 515 294-7212

Abstract: Parallelization Agent (PA) is a knowledge-based system forparallelizing 3-dimensional, time marching, explicit finite difference science codes. It The MM5, RAMS, RADM and a couple of other codes have been parallelized with help of the PA. It was possible to parallelize the MM5 code in two weeks as opposed to several years it took for manual parallelization by a team of expert programmers.The PA provides a customized but flexible environment forparallelizing atmospheric science codes. It supports portable higher level abstractions that simplify the process of parallel programming by automating tedious details of parallel programming and hiding low level architectural details. The PA has an interactive capability for exchanging information about specific applications. This interaction is closer to the way application program methods are understood and communicated by humans.

1. Introduction

Parallel computing can provide the computational resources to perform large-scale simulations needed in atmospheric and environmental sciences. Johnson et al [1] report results of a regional prediction model run on a parallel computer, and Kim et al [2] discuss speed-up and load balance for an implementation of the Penn State/NCAR MM5 model on 64 processors. However, there are major hurdles in applying parallel computing to run legacy codes in atmospheric sciences. Manualparallelization is tedious, prone to errors, and very time consuming.

Difficulties in manualparallelization point to a need for automation. Several automatic and semiautomatic tools have been developed. Doreen Cheng has published an extensive survey [3] with 94 entries for parallel programming tools out of which 9 are identified as "parallelization tools to assist in converting a sequential program to a parallel program." A full automation of parallelization process requires solutions to problems that are known to be NP-complete (intractable in a technical sense). The emphasis of recent research has been on developing interactive tools requiring assistance from the user.

The Parallelization Agent (PA) [4,5], developed at Iowa State University, is a tool for developing efficient parallel programs based on the message-passing programming model. As opposed to parallelization of arbitrary programs, the PA is centered on a key numerical method to provide a customized but flexible environment for building parallel physical science simulation codes. We have used the PA toparallelize several atmospheric codes. At the Athens conference, we demonstrated parallelization of MM5 and RAMS using the PA. Using MM5 as the example, this paper describes capabilities of the PA and demonstrates how it is used.

2. Parallelization Agent (PA)

So far, the research is predominantly focused on automatic parallelization of arbitrary sequential programs. The PA shifts the focus to specific classes of codes that are based on numerical methods for modeling a broad range of physical phenomena. By focusing on a class of problems, the class-specific high-level knowledge can be used to simplify the otherwise intractable problem of automatic parallelization. Historically, a similar shift in focus has occurred in the domain of artificial intelligence when expert systems were introduced. While the problem of developing an *intelligent program* is too difficult in general, the idea of expert systems has proved to be fruitful in addressing important problems of special interest.

The PA development takes a pragmatic approach by automating tedious and time consuming tasks rather than striving towards complete automation. For example, it is very difficult to automate recognition of the algorithmic form. The knowledge about the algorithmic form is crucial for processing complex codes. A programmer, for instance, will first find out that the model is based on the *finite difference method* (FDM) and then use knowledge about the FDM to proceed with parallelization. The PA follows a similar process. To identify the underlying numerical method, it relies on the user.

The programming model is the *single program multiple data* (SPMD) model with message passing for inter-processor communication. The current version of the PA handles FORTRAN 77 codes. The PA employs a structured process that relies on specific knowledge about the numerical method to arrive at an efficient parallel program. The PA currently supports parallelization of 3-dimensional time marching explicit finite difference codes. The PA system runs on Unix workstations and PCs running under the Linux operating system.

2.1 Capabilities

The PA is envisioned as a part of the computational infrastructure to enable advances in physical science simulation problems. The PA can be useful in multiple ways. In addition to parallelization, it can be used to diagnose serial code or to display different type information about serial or parallel codes. It can assist application scientists in discovering important information about a code without having to go through it line by line. Atmospheric scientists will find the following capabilities useful:

Diagnostic facility

Each individual source file can be diagnosed. The PA can point out various problems such as: data exchange patterns not consistent with the differencing scheme, ambiguous uses for loop indices. The PA provides auxiliary information and specific references in the code to resolve these problems. The auxiliary information, for example, will show the indexing patterns that are causing the ambiguity. The diagnostic facility is interactive and allows exchange of information between the user and the PA. The PA indicates if a

source file is ready for parallelization. If not, the user has to respond to the problems identified by the PA.

Display of call-order tree

The PA traces the sequence of subroutine calls and displays the *call-order tree*. This helps the user to understand the structure of the code. After parallelization, the call-order tree includes markings to show the subroutines that will have communication.

Selective Parallelization

The user can parallelize either the selected subroutine only or all routines that are called from the selected routine. The selective process allows the user to view the effects of including subroutine calls. The PA does inter-procedural analysis and shows additional communication that may be introduced due to subroutine calls.

Display of Code Blocks

Parallelization results in breaking the given code into separate blocks with synch/exchange points between successive blocks. The PA shows the sequence of blocks. The user can click on a block to see the serial code corresponding to that block. A display of code blocks helps the user to understand the relationship between the serial and parallel codes.

Display of Communication Stencils

The PA determines the data exchange patterns at each of the synch-exchange points. The user can view these patterns in the form of stencils showing the communication that will occur in parallel processing. Application scientists find these stencils particularly valuable because they also show the differencing scheme at work.

Automatic Parallelization

The diagnostic phase requires assistance from the user and may take a couple of weeks for a large code like MM5. However, the parallelization itself is automatic and quick. The PA does global-to-local index transformations necessary to convert the serial code into a parallel code. It identifies the communication requirements, optimizes communication, and inserts the message passing primitives in appropriate places. The PA allows three options for generating the actual parallel code: run-time system library, MPI, or PVM. At present, the first option is operational.

Graphical Interface

The user can interact with the PA system through a graphical interface. By click of a button the user can select a file, diagnose source code, view the call-order tree, parallelize selected files etc.

2.2 Using the PA

In this section we describe the mechanics of using the PA. The steps are as follows:

1. As observed earlier, the PA makes use of class-specific information to process the code. The user has to first identify the class of the code to be processed. This is done by selecting one class from the choices given by the PA. For MM5, RAMS and other codes based on the finite difference method, the user needs to select the FDM class.
2. Next the user gives the pathname for the code directory, identifies the key indices from the specific code, and provides a high-level parallel mapping. The interface for providing this information is shown in Figure 1.
3. The user can first view the call-order tree for the specified code and decide on the routines to be processed.
4. The user must diagnose the selected routines before they can beparallelized. To diagnose a routine the user selects the routine from a menu.
5. If a selected routine has a problem, the PA displays the problem on a screen. To resolve the problem, the user may have to either provide auxiliary information about certain array variables or make changes to the serial code. The PA provides specific references to locations in the code and other information to assist the user in resolving the problems.
6. After an attempt to resolve the problem, the user can diagnose the selected routine again and the PA indicates if the routine is ready forparallelization. If the user tries to parallelize a routine before resolving the problems, then theparallelization fails. The current version of PA indicates that the parallelization failed but does not provide any additional information. In this situation the user needs to go back to the diagnostic phase.
7. Once all the selected routines are diagnosed the user can proceed to the parallelization phase. The user can choose to parallelize only the selected routine or all the routines called by it. The parallelization is automatic. The call-order tree is traversed from bottom up during parallelization. The routines at the lowest level of the call-order tree are processed first and the routine at the root is processed last.
8. The user can identify the routines where communication occurs by viewing the call-order tree. The PA marks these routines during parallelization.
9. The user can check where the synch-exchange points are inserted by invoking the PA to display the code blocks. The user can click on a block to see the serial code corresponding to that block.
10. The user can view the communication at a synch-exchange point by selecting the synch exchange point and clicking the stencil button.
11. Finally, the user can generate the parallel code by clicking a button. The PA displays three options: run-time system library, MPI, and PVM. The first option is operational at present.

3. Parallelization of MM5

As an experiment, we used the PA to parallelize MM5. The *solve1.f* and all the underlying subroutines wereparallelized. The parallel code generated by the PA was

compared with the code developed by expert programmers. Here we describe our experience of parallelizing the MM5 code using the PA. To provide a concrete illustration, actual displays from the PA are reproduced.

To start the process, we identified MM5 as a finite difference code. We wanted the parallel MM5 to map the three-dimensional domain onto a two-dimensional array of processors so that the computations in a column of nodes are assigned to a single processor. The same mapping is used in manualparallelization [2,6]. The MM5 code uses indexes i, j along the horizontal plane and k along the vertical direction. To direct the PA to follow the desired mapping, the user needs to identify i and j as the indices for parallelization. The directive is conveyed to the PA, as shown in Figure 1.

The call-order tree for MM5 is shown in Figure 2 C. The call-order tree was used to identify the subroutines called by *solve1.f*. These subroutines were diagnosed. It took two weeks for one person to complete the diagnostic phase.

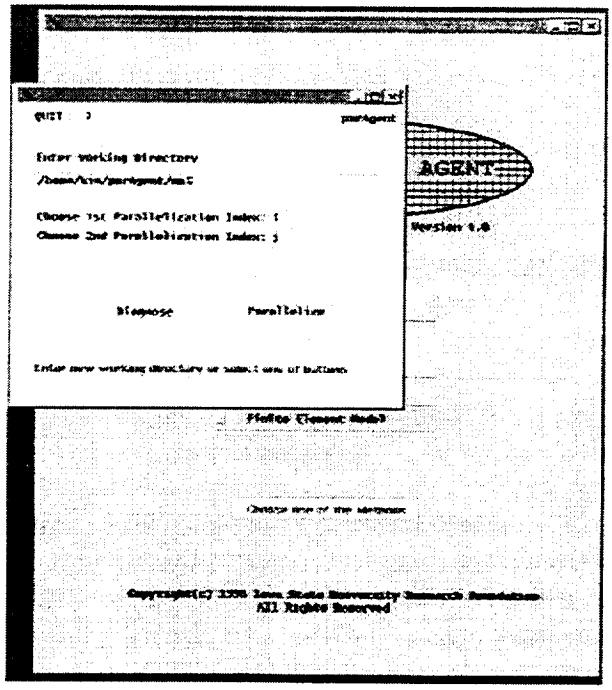
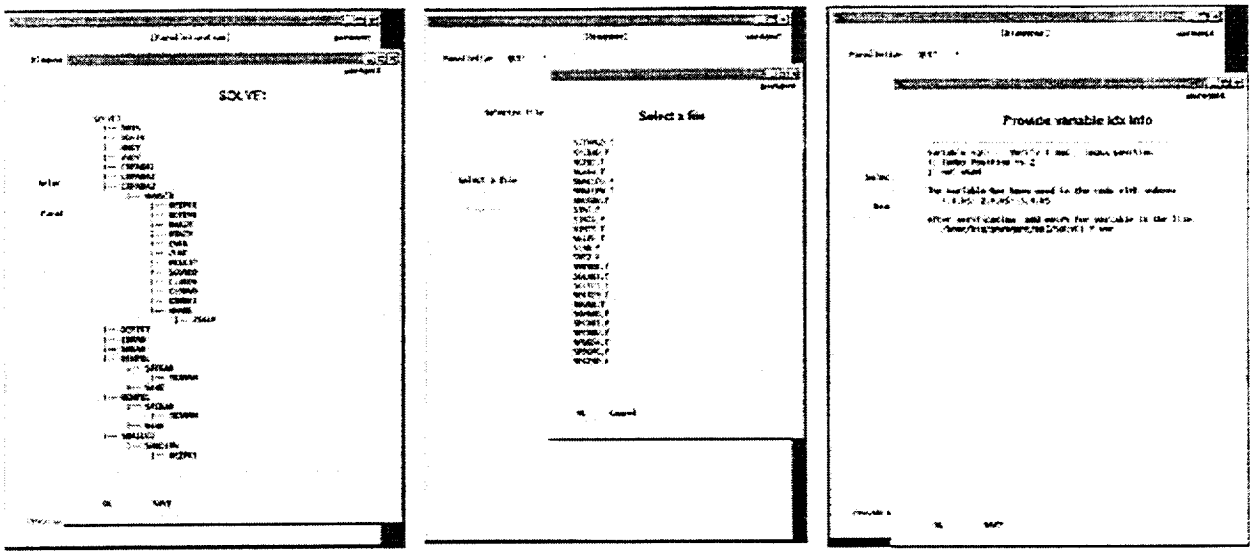


Figure 1: Initial information given to the PA

We observed that the problems detected during this phase arose from two main sources: aliased array variables and ambiguous or inconsistent indexing. For example U1(j , k) is used as an alias for U(1, j , k). One situation for ambiguity is that an array is indexed by constants and it is not clear from the context if the array is supposed to be indexed by i or j. The displays for call-order tree, selecting a file for diagnosis, and diagnostic information from the PA are shown in Figure 2.



a. Call order tree

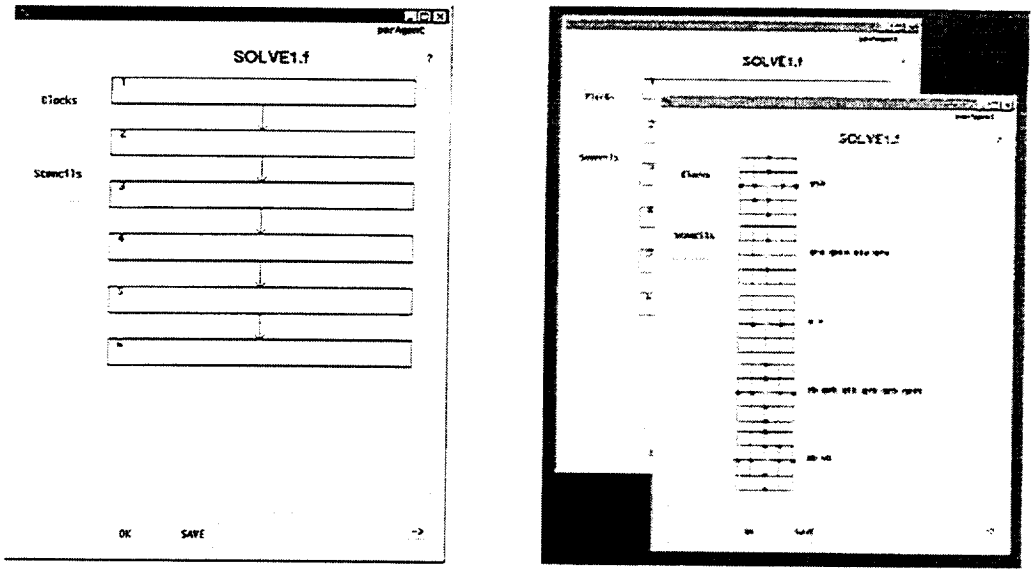
b. Selecting a file for diagnosis

c. Diagnostic information

Figure 2: Diagnostic Phase of the Parallelization Agent

After the routines are diagnosed, the next step is parallelization. Theparallelization is automatic and very fast. For *solve1.f* and all its subroutines, it takes a minute or so. During this process, the PA determines the synch/exchange points and the inter-processor communication at each of the synch/exchange points. The results ofparallelization can be viewed through a graphical interface. As shown in Figure 3a, theparallelization of *solve1.f* resulted in six blocks of code separated by synch/exchange points. The serial code corresponding to a block can be viewed by clicking on that block. The stencil for one of the communication points is shown in Figure 3b. The stencils are observed by clicking the stencil button and selecting a synch/exchange point.

The final step is generating a parallel code. This step involves global-to-local index transformation to modify loops in the serial code and insertion of communication primitives. Prior to this stage, the PA has determined the communication pattern, and the objective at this stage is to take care of the lower-level details of communication. The PA is designed to support different alternatives including MPI for handling the details of communication. Currently, the PA supports a modified version of the run-time system library [7] developed at Argonne National Laboratory.



a. Parallel code blocks **b. Stencils showing communication**

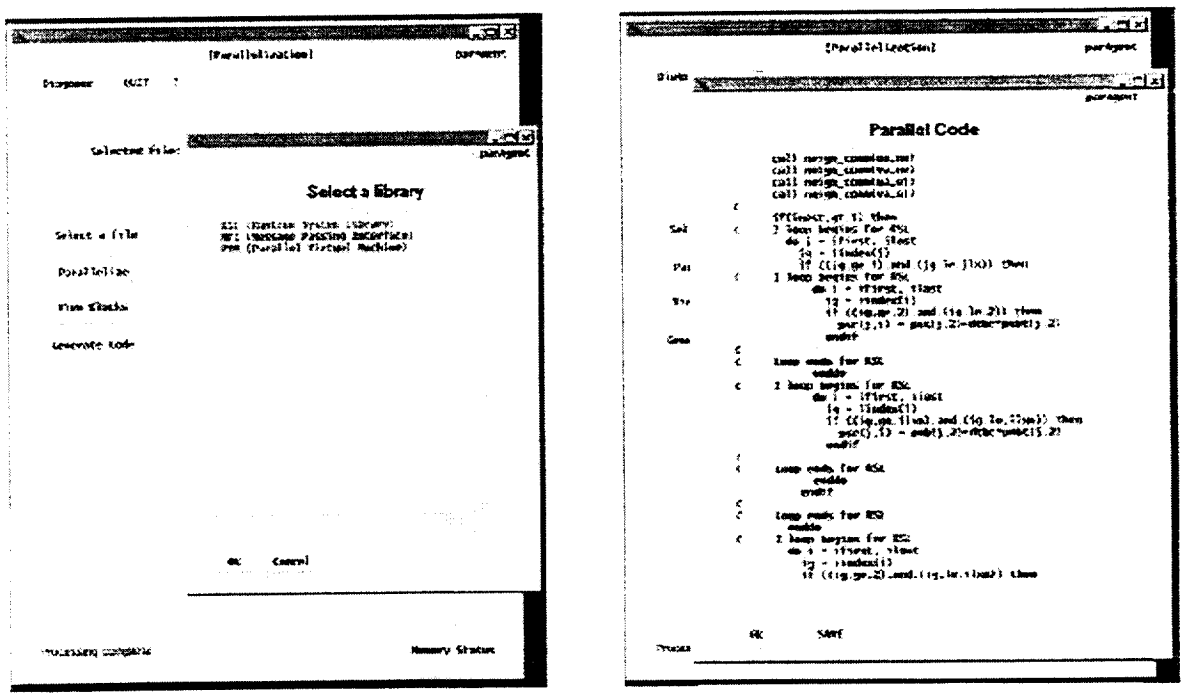
Figure 3: PA Displays showing information about parallel MM5

3.1 Observations

- The parallel code produced by the PA has six synch/exchange points as opposed to three for the code developed by expert programmers [2, 6]. Without any optimization the code has hundreds of synch/exchange points. Thus, the PA came very close to human experts in minimizing the synch/exchange points. Moreover, we found that it is not too time consuming to do the further reduction by hand to get to from six to three synch/exchange points.

- The PA approach is very helpful in eliminating debugging of parallel code. The manual process is prone to errors. In a large and complex code such as MM5 that involves hundreds of variables, more that hundred source code files and thousands of lines of code, debugging the parallel code is very difficult. The PA uses a formal and automated method that eliminates errors inherent in a manual approach. However, one must ensure that the PA is given correct information when it detects a problem and asks for additional information during the diagnostic phase. We obtained some of this information by talking to the application scientist. As opposed to spending effort on debugging parallel code, the PA approach is more amenable to high-level reasoning and leads to better understanding of the code that is likely to be useful for maintaining the code in the long run.

- The PA dramatically reduced the amount of time and effort. It took a team of programmers more than three years to complete the parallelization of non-hydrostatic version of MM5. With the help of the PA, a single person completed the parallelization of the MM5 code in two weeks.



a. Selecting a communication library

b. Parallel code

Figure 4: Generation of parallel code

4. Conclusions

The Parallelization enables efficient parallelization of legacy codes in atmospheric sciences. It automates tedious and time-consuming parts of the parallelization process. It eliminates commonly encountered errors in the manual parallelization of large and complex codes. It supports portability by producing parallel codes that can run on a variety of distributed memory platforms. The PA can produce scalable parallel codes for massively parallel computing. The PA approach will facilitate management and evolution of complex codes. Often parallel models cannot keep pace with advances in the serial model because of the prohibitively long time it takes to parallelize serial models. The PA will be an effective tool to address this problem.

The PA can be enhanced to extend its applicability and usefulness. The PA is based on class-specific approach and it can be enhanced to incorporate other classes. Currently efforts are underway to create similar tool for codes based on the finite element method. There is also a need for providing an affordable parallel solution. It will help many countries to take advantage of advances in atmospheric sciences to plan growth and manage resources. With the advent of powerful and relatively inexpensive PCs, a cluster of PCs can be a reasonably good medium for parallel computing. We are currently experimenting with the PA on a cluster of PCs.

References

1. Johnson, K., J. Bauer, G. Riccardi, K. Droegemeier, and X. Xue, *Distributed processing of a regional prediction model*, Mon. Wea. Rev.,122, 2558-2572, 1994.
2. Youngtae Kim, Zaitao Pan, Eugene S. Takle, and Suraj C. Kothari, *Parallel Implementation of the Hydrostatic MM5*, 8th SIAM Conference on Parallel Processing for Scientific Computing, 1997.
3. Cheng, *A Survey of Parallel Programming Languages and Tools*, Tech. Rep. RND-93-005, NASA Ames research center, Moffet Field CA 94035, 1993.
4. Simanta Mitra, Suraj C. Kothari, *Parallelization Agent: A New Approach to Parallelization of Legacy Codes*, Eighth SIAM Conference on Parallel Processing for Scientific Computing, March 1997.
5. Simanta Mitra, *Parallelization Agent: A New Approach to Parallelization of Legacy Codes*, Ph.D. thesis, Computer Science Department, Iowa State University, December 1997.
6. Michalakes, J., 1997b: MM90: A Scalable Parallel Implementation of the Penn State/NCAR Mesoscale Model (MM5), Parallel Computing (to appear); also preprint ANL/MCS- P659-0597.
7. Michalakes J., *RSL: A parallel runtime system library for regular grid finite difference models using multiple nests*, Tech. Rep. ANL/MCS-TM-197, MCS Division, Argonne National Laboratory, Argonne, Illinois, 1994.
8. http://www.cs.iastate.edu/~hpc: Information on Parallelization Agent

DEEP: A Development Environment for Parallel Programs

Brian Q. Brode and Chris R. Warber
Pacific-Sierra Research Corporation
2901 28th Street, Santa Monica, CA 90405
info@psrv.com
+1 310 314-2300

Abstract

The use of the DEEP development environment to analyze parallel program performance is described. The full integrated environment contains tools for the creation, analysis and debugging of parallel programs. All information is related back to the original parallel source code. This paper describes the program analysis portion of DEEP and describes its use on parallel programs with distributed memory (message passing with MPI, data parallel with HPF and Data Parallel C), and shared memory (automatic parallelization and OpenMP).

1. Introduction

The DEEP system provides an integrated parallel program development environment that binds debugging and performance tools back to the original parallel source code. DEEP includes many useful tools in a highly interactive integrated GUI interface, and it provides a simple and intuitive way to understand and investigate parallel program structure, performance, and behavior.

The goal of the system is to support parallel programming for the most frequently used languages and target systems, in the same framework. To this end, DEEP supports both distributed and shared memory parallel programming, and both Fortran and C languages, within a single DEEP executable. Having a single system that supports all of these parallel programming models and languages provides great flexibility for the user. Most parallel programmers should benefit from the system.

1.1 Distributed Memory Programming.

Distributed memory parallel computer systems can range from networks of workstations or even PCs to large-scale massively parallel computer systems designed for efficient inter-processor communication. Programming for distributed memory systems is often done with explicit message passing systems (such as MPI) or data parallel languages (such as HPF or Data Parallel C). DEEP supports both of these programming methods.

1.1.1 MPI-2 Support

The MPI-2 message-passing interface [1] allows the user to write a portable distributed memory program with a rich set of calls to allow the interchange of information between processors.

DEEP keeps track of all MPI-2 calls in a Fortran or C program, and highlights and profiles these in the program analysis tools; this allows a better understanding of the performance characteristics of MPI codes. Debugging of MPI-2 programs can be a challenge, as several groups of processors can be working on different calculations in parallel; the DEEP debugging interface allows the user to see what is happening in each parallel group.

1.1.2 Data Parallel Language Support

Data parallel programming allows the user to concentrate on the higher level aspects of the decomposition of the program data across the available processors, while the compilation system performs all the low-level bookkeeping and provides for all of the communication needed between processors. Data parallel programming languages include High Performance Fortran[2], which is an emerging standard first proposed in 1993, and the Data Parallel C Extensions (DPC)[3], accepted as a technical report in 1994 by the X3J11 ANSI C Committee. DEEP works with both HPF and DPC programs[4], and can even support mixed language Fortran/C applications

Data parallel languages have two distinguishing features: syntax for describing the distribution of data across processors, and a method for making clear the parallel nature of calculations. HPF has a set of directives that allow specification of data distribution and DPC has "shape" declarations for this purpose. Both HPF and DPC have array syntax, which allows entire arrays of data to be manipulated with single statements.

Critical to the performance of data parallel codes is whether calculations are successfully partitioned across processors, and where the data parallel compiler had to add message-passing code. The DEEP system provides feedback to the user so that performance problems can be quickly identified.

When debugging of data parallel programs, the user would like to treat the program as a single executable, and the DEEP debugging interface makes this possible.

1.2 Shared Memory Programming

For shared memory systems, DEEP supports Fortran and C programs that are being automatically parallelized[5], and programs that are being parallelized by hand with the recently specified OpenMP[6] set of directives. DEEP for SMP targets systems that support threads, such as the POSIX threads standard, and the system works on both UNIX systems and Windows/NT.

1.2.1 Automatic Parallelization Support

With DEEP, users of automatic parallelization can quickly see what loops have been parallelized and what loop nests have been put into parallel regions. This allows the user insight into the compiler's actions and the performance that results.

1.2.2 OpenMP Support

OpenMP is a large set of directives that allow the user to direct parallelization of a code. DEEP keeps track of OpenMP constructs and profiles the execution of OpenMP codes so that the user can see the effect of using the directives on the performance of the application.

1.3 Using DEEP

DEEP program analysis tools allow use of both compile-time information (gathered by the compilers) and run-time information (gathered by a run-time profiling library) to investigate a parallel program in more detail. In analyzing a program with the DEEP system you would normally start with the tools that look at the whole program, identify procedures of interest, and then "drill down" with tools that look at the internal structure of individual procedures.

2. The DEEP Framework

The DEEP system generates an abundance of information about a parallel program. Presenting this type of information in a coordinated package can be a significant challenge to user interface design. DEEP is organized around a single GUI framework that organizes and displays the wealth of information provided by the system. The DEEP framework is organized into user-configurable panels that contain "viewers". Each viewer in turn contains pages that hold the information.

Panels can be reconfigured by grabbing and moving. Normally, a user will have three to five panels open. When the size of one panel is changed, the other panels automatically

readjust themselves accordingly. We find this arrangement much easier to deal with than many independent windows popping up all over the screen. Also, a panel can be expanded to fill the whole DEEP window, then restored to its previous configuration; this is easy to do through a right-click pop-up menu.

Within these panels are viewers, a viewer being a software tool for examining some aspect of a parallel program. Viewers can be docked in any panel or popped into their own window. Viewers in a panel or pages in a viewer are selected with a click of a button. Figure 1 shows a DEEP display with three toolbars (along the top) and four panels. Tabs along the bottom of the panels select viewers inside the panel; tabs at the top of each panel select between pages in a viewer.

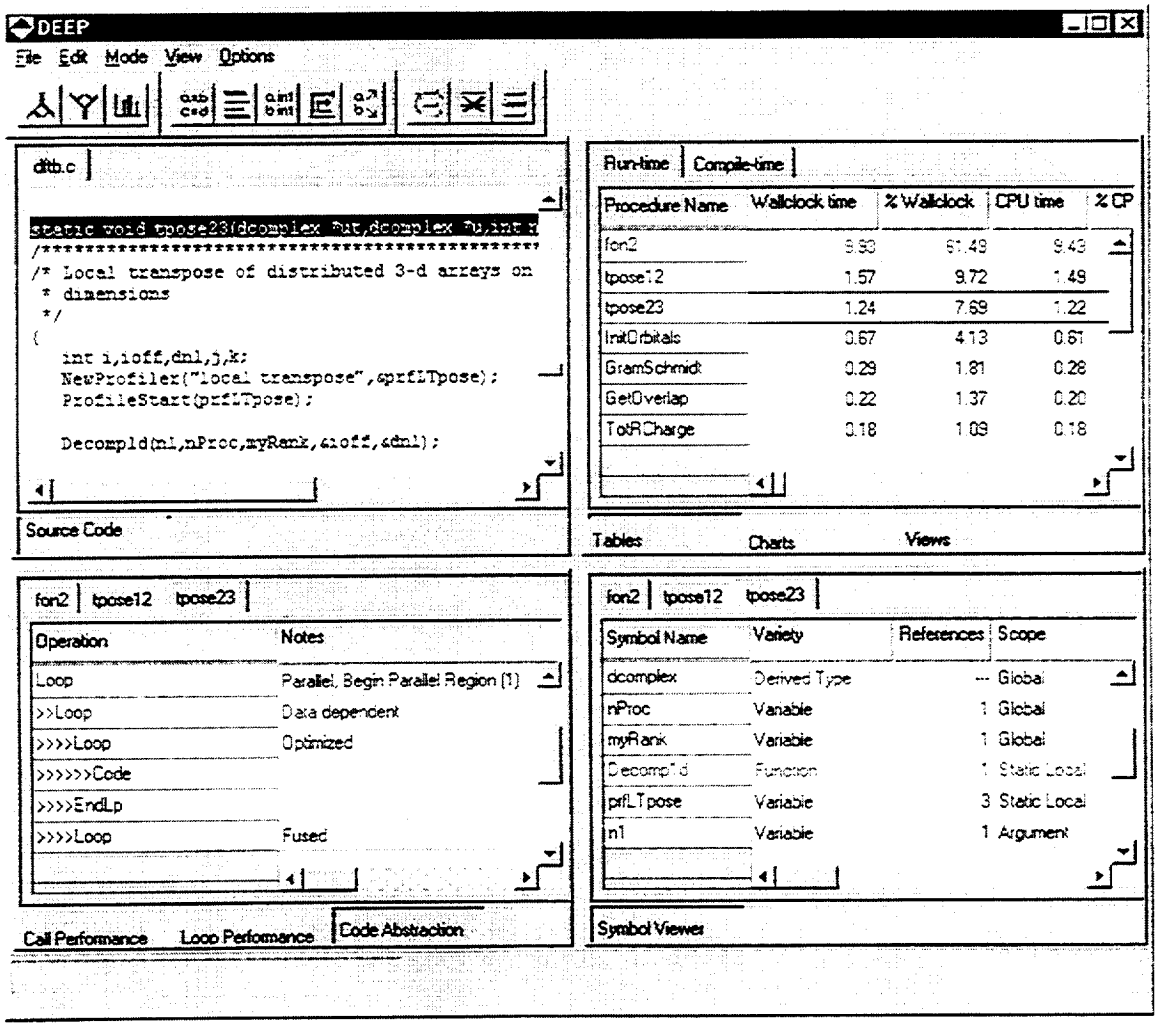


Figure 1: DEEP Framework

Three toolbars can be seen near the top of Figure 1. Toolbars contain buttons that invoke various functions. The buttons contain help tips, which is useful if the icon does not clearly bring to mind the function to be performed. The toolbars can be dragged to docking areas at the top, bottom, left and right of the panels.

3. DEEP Program Views

This section examines the DEEP program analysis tools that display some aspects of the whole program. These tools are the starting point for understand program structure and performance.

3.1 Program Information Table

DEEP allows you to inspect the program as a whole in various forms. For inspection of raw data gathered from compiling and executing the program, the *program information tables* are the place to start; they list each of the procedures in the program with various statistics. The statistics display are customized based on the target system -- different statistics are displayed for an MPI code than for an OpenMP code, for example. For an MPI program, statistics about message-passing are paramount; for OpenMP, statistics about parallel regions are important.

There is a compile-time table that shows information gathered from compile-time analysis and optimization of the program, such as the number of parallel loops. The run-time information table lists CPU time, wallclock time, number of messages passed, number of calls, number of loops executed, average iteration count, etc. There is also an inclusive runtime table which has data for the indicated routine and all routines it calls. Finally, there is a loop table that presents all of the instrumented loops in the program. The tables can be sorted on any field, and procedures of extra interest can be highlighted by user-selected criteria. Procedures that are not of interest can be pruned. Selecting a procedure in the table (by clicking with the mouse) brings up detailed information about the procedure in other panels.

The buttons in the tool bars at the top of the display can be used to launch other views of the program as well. The example in the upper right quadrant of Figure 1 shows three of the time-related fields in the program table; by default, the table is sorted by wallclock time and all procedures that use more than 10% of the wallclock time are highlighted.

3.2 Call Tree Viewer

The *call tree display* allows browsing of the call relationships of the procedures. Any procedure can be set as the root, and both calling trees (down to leaf procedures) and called trees (back to the main procedure) can be browsed. If runtime data is available, the trees are annotated with inclusive time and (for distributed memory targets) message counts. This performance annotation helps direct the browsing of the tree to branches that used the most resources.

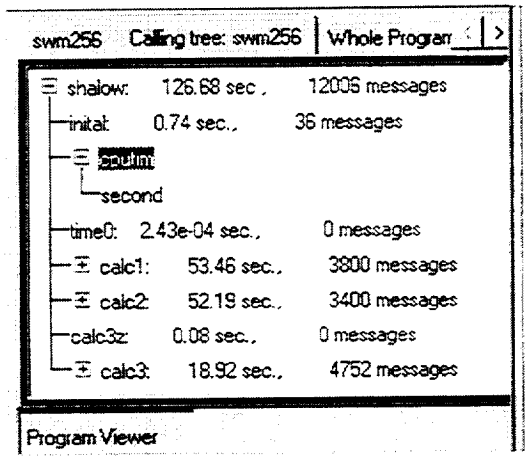


Figure 2: Call Tree View

The tree can be expanded or contracted at each level by clicking on the + and - icons (see Figure 3). When you get to a point in the tree where you would like to have more information, you can use the tool bars to move directly to detailed displays, or the source code, of the selected procedure.

3.3 Whole Program Viewer

With the *whole program viewer*, the entire program can be examined on one screen, with color-coded lines of pixels representing performance of source lines in each subprogram. Each procedure is represented by a rectangle which is proportional to the number of source code lines. The lines of pixels represent the individual lines of code, and are indented to correspond to control structures in the original source (loops, conditional blocks, etc.). Lines can be color-coded based on compile-time optimization feedback (for example partitioned loops in blue, non-partitioned loops red) or on run-time instrumentation (lines colored coded based on the amount of time spent there).

Clicking on any pixel in a rectangle brings up the corresponding source line in the source code editor, and also establishes that area in the code abstract viewer for that procedure.

This allows you to move quickly from the high-level program information to specific information about an area of the program.

In Figure 3, DEEP uses static performance information from a data parallel compiler to color-code lines in each procedure, based on how well they were compiled for parallel execution. Source lines that result in inefficient code (messages need to be passed to other processors) are highlighted at the red end of the spectrum; source lines of loops that are partitioned cleanly (with minimal message passing) across the parallel processors are highlighted at the blue end of the spectrum (see the key at the top of the window in Figure 4). Other colorings of the whole program display are possible, including ones based on dynamic performance information.

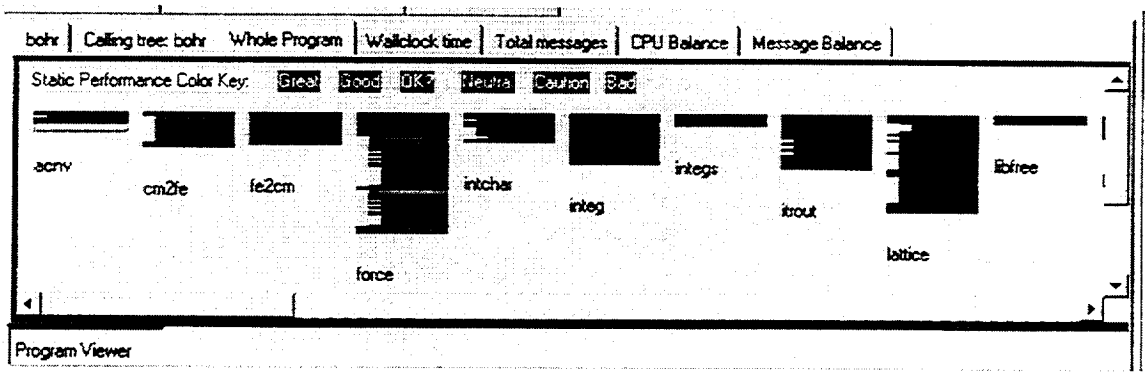


Figure 3: Whole Program View

3.4 Summary Charts

Several summary charts are prepared, based on the target parallel system. The charts can include the wall clock time by procedure and the breakdown of total messages passed by procedure. These let the user quickly see where the resources are being spent in the parallel program. These charts are presented as colored pie charts or bar graphs.

3. 5 Load Balance Displays

The *message load balance display* and the *CPU load balance display* are intended to give information on how the computational load is distributed. Is one processor sending most of the messages or doing most of the processing? If so, then you may want to reconsider how you have distributed the data. For instance, a block-cyclic approach may provide better load balance than a pure block distribution on a data parallel program. For SMP programs, low utilization of the last few threads may indicate that the program can get by with less threads.

An example of a message balance display for a data parallel program run on four processes in seen in Figure 4. The display shows the number of sends and receives for each logical process.

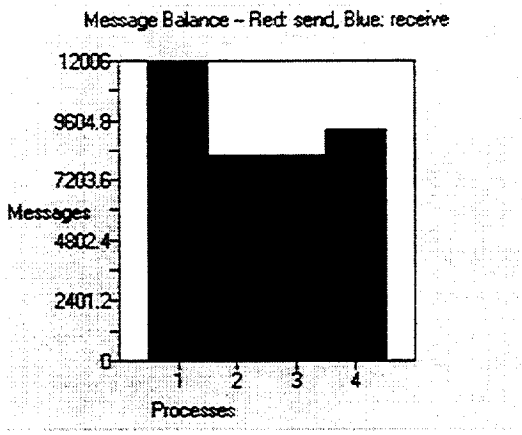


Figure 4 – Message Load Balance

4. DEEP Procedure Views

When a procedure of interest is determined by examining the program-level views, you can move to the more detailed procedure-level displays.

4.1 Code Abstract Viewer

The *code abstract viewer* allows the programmer to examine routines with an overview of important control structures. This view is annotated with optimization notes from the data parallel or SMP compiler. From this display, you can move to the corresponding line in the source code editor with a click of the mouse. The display is color-coded to provide performance feedback at a glance. The lower left quadrant of Figure 1 shows part of the abstract for a routine. OpenMP directives and MPI calls are preserved in the Code Abstract View.

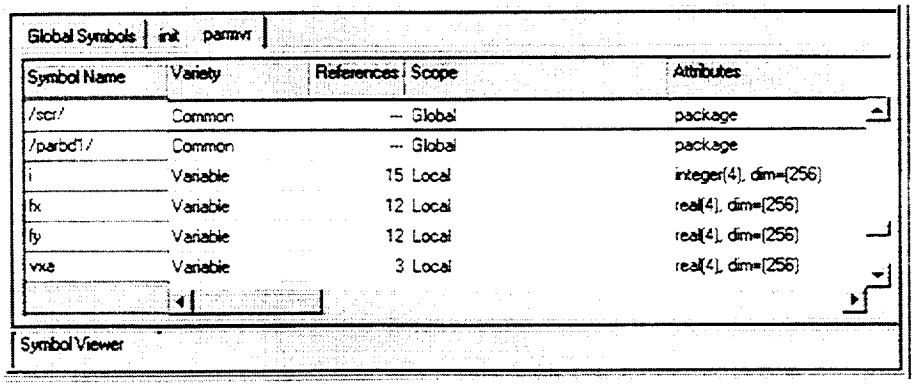


Figure 6: SymbolViewer

4.2 Symbol Viewer

You can track the uses and definitions of variables throughout the parallel program with the *symbol viewer*. In the lower right quadrant of Figure 1 various kinds of symbols can be seen. Symbol information including scope and attributes is available. The file of definition is displayed, and if clicked on will bring up this file in the editor.

The *global symbol page* lets you see where global symbols are set and used across procedures and files. This can be very useful in large programs.

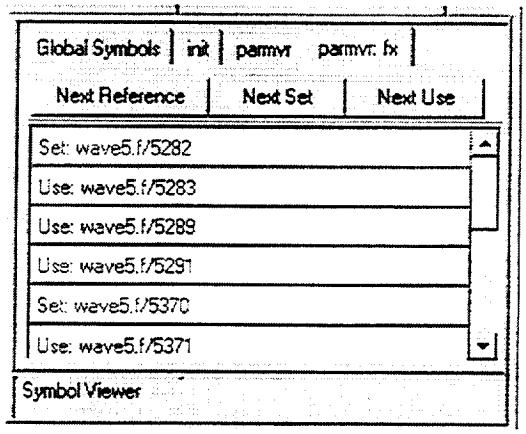


Figure 7 – Symbol References

Clicking on the "References" column brings up a new page that lists all references of a variable, and allows the user to move through these references with the click of a button. As each reference is selected, the corresponding line of code is displayed in the source code editor.

4.3 Performance Viewers

The *loop performance viewer* and *call performance viewer* present tabular information on the details of a procedure's performance. This performance information is gathered during run-time at the individual loop and call level. The loop performance viewer combines compile-time loop optimization information with measured run-time performance.

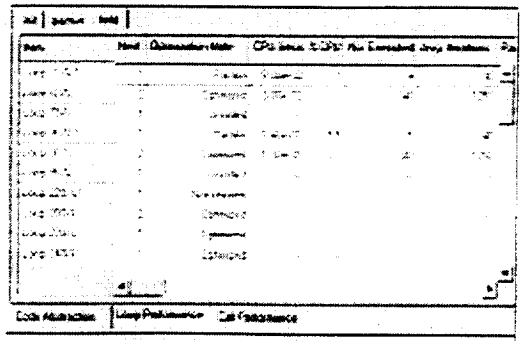


Figure 8 – Loop Table

For MPI-2, there is an *MPI viewer* that summarizes information about the execution of MPI calls in the current procedure.

5. DEEP Debugging Tools

DEEP also provides run-time interactive GUI-based debugging of parallel programs at the source code level. A demonstration screen snapshot of the debugger interface can be seen in Figure 9. DEEP's debugger was designed so that the programmer can debug the source code at the level of abstraction most natural for the type of parallel program. Tools include:

- *Source Code Viewer.* Shows the execution location in the source code. Features language-sensitive syntax highlighting, and interactive symbol browsing and variable value inspection.
- *Inspection, Watch and Locals viewers.* DEEP provides several viewers that display the value of variables. Simple variables are best displayed in Watch viewers, and structured variables in Inspection viewers. The Locals viewer is similar to the Watch viewer, showing all active variables in their context. The variables to be watched or inspected can be selected by dragging from any other viewer, such as the Source Code Viewer or the Symbol Browser.

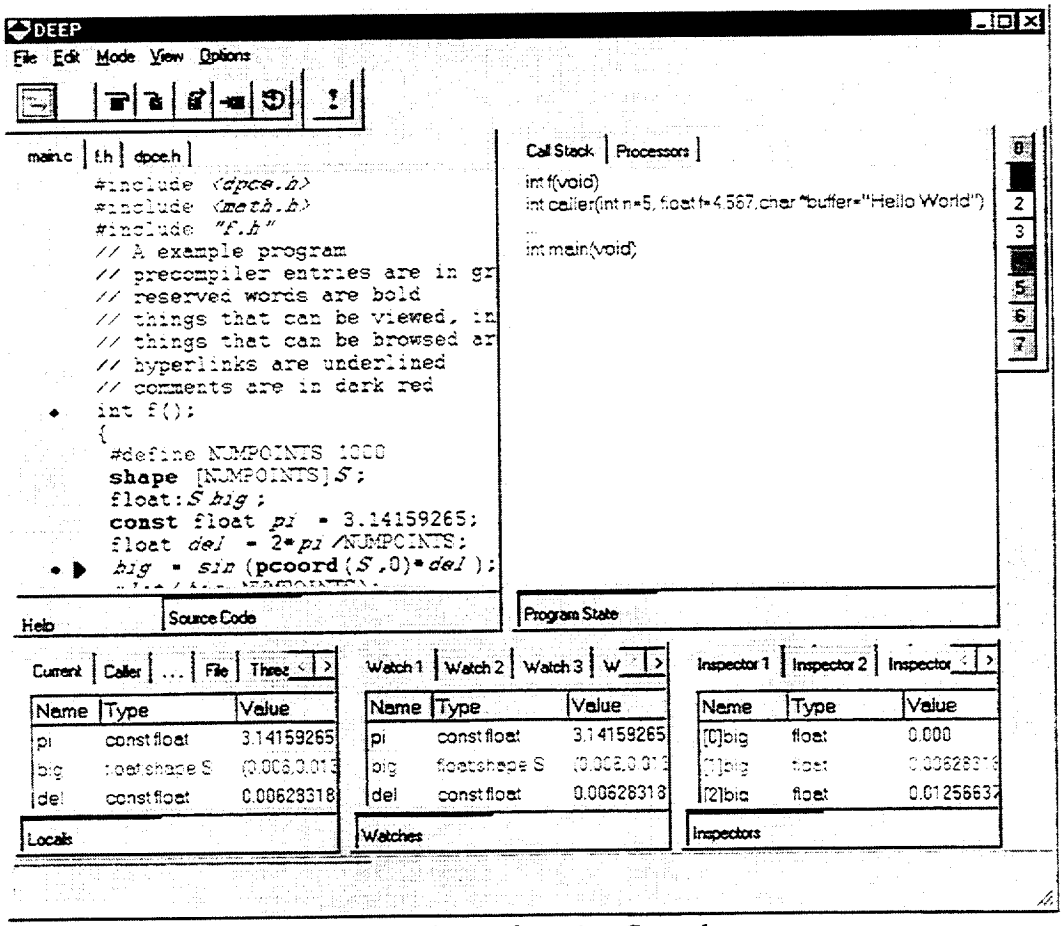


Figure 9 – Debugging Snapshot

- *Graphical array viewers.* Multidimensional distributed arrays can be visualized using two and three-dimensional graphical tools.
- *Breakpoints.* Both conditional and unconditional breakpoints are supported. Breakpoints can be set with a single click.
- *Watch points.* Watch points suspend execution when the value of a variable changes.
- *Trace points.* When a trace point is reached in the source code, user selected expressions are displayed in the program log viewer.
- *Performance Zones.* You can establish performance zones in the source; the system can display the CPU time, wall clock time, messages sent or received, and I/O done by each process in each zone.
- *Processor Status.* The current state of all the processors is shown by its color in the processor status viewer. In addition, detailed information about individual processes can also be displayed.

In order to support different types of parallel programs DEEP uses the concept of a *processor set.* A processor set can contain all processors that are running the same executable. Users are free to define their own processor sets, and a set can contain only one processor. DEEP's viewers display information gathered from the current processor set. For example, since Data Parallel programs are normally viewed as a single object, the most natural processor set contains all the processors. Thus, the programmer can view the code in as if it were a single executable. Arrays that might be distributed over a number of processors are treated by the debugger as single object. The de-bugger handles the tasks of assembling the parts of array from the individual processors when the programmer views or manipulates values in the array. The debugger makes sure that the processor that make up a set are kept in synchronization when breakpoints are triggered and during single stepping.

Processor sets are created by the DEEP debugger according to the type of parallel program. The user can modify DEEP's sets and create other sets by dragging processor icons into set icons. The user can also create toolbars that show the state of the sets or of individual processors. In Figure 9, DEEP is being used to debug a data parallel program running on eight processors. The toolbar along the right side of the display uses colors to display the state of each processor within the single processor set. If the debugging session contained other processor sets, the user has the choice of displaying a toolbar for each set. In addition, a summary toolbar showing one icon for each set can be displayed.

Breakpoints, trace points, etc. are set on all the processors in the current processor set. The easiest way to set a breakpoint is to right click in the breakpoint area (the gray left border of the source code viewer shown in Figure 9) and selecting the desired type from the popup menu displayed. Figure 10 shows the dialog shown to set a conditional breakpoint.

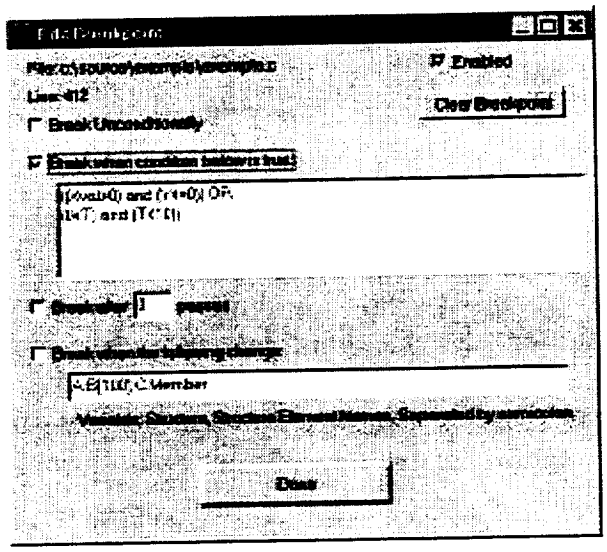


Figure 10, Edit Breakpoint Dialog.

Performance zones are designed to allow the user to time areas of the code. A performance zone is made up of two performance points. The performance point that starts the zone, called the start point, is denoted by ▼ in the breakpoint area, and the point that ends the zone, the stop point, is denoted by ▲. The user ties a pair of points together by giving each the same name. The user may assign colors to the performance zone.

When the user code reaches a performance point messages may be logged in the message log viewer and the program time line viewer. The color of the performance point tags the message display. If the point is the start point of a zone then the following timers or counters may be started:

- The CPU timer.
- Message received counter.
- Message sent counter.
- I/O counter.

The user code then continues execution. If the point is the stop point of a zone, the timer and or counters are stopped and appropriated messages can be sent to the message log and/or the time line.

Figure 11 shows the dialog box used to define a performance point in a C program. In this figure, we are also showing the select color dialog box.

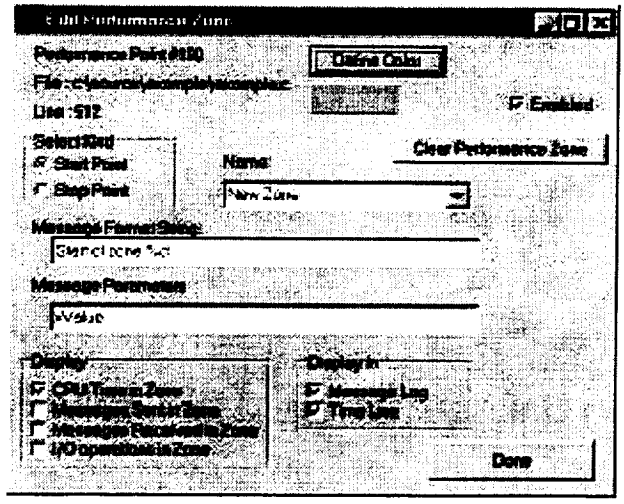


Figure 11. Performance Point Editing Dialog, with the color selection dialog.

The source code viewer is fully integrated with the rest of the debugger. In the Figure 12, the right mouse button has been clicked over a variable name, which cause a pop up menu to appear with details about the variable. Also shown in the figure is a blue triangle which indicates the next source code line to be executed, and the pink bar shows where a temporary breakpoint has been set.

Variable names can be dragged from the source code viewer onto other viewers where appropriate. For example, if a name is dropped on to a watch or inspector window, the value of the variable will be displayed during execution. Names can also be dropped onto the browser window, which will cause the browser to display its information about the variable.

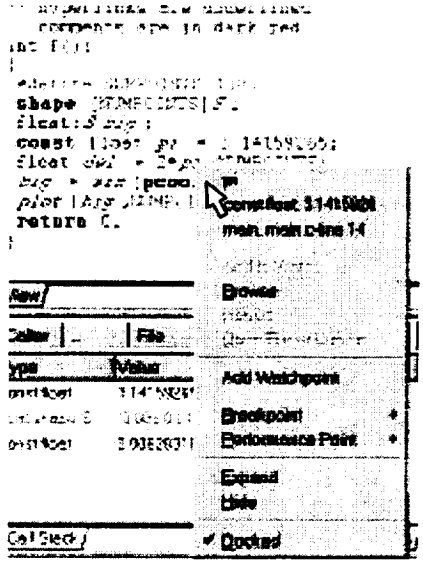


Figure 12. Source Code Browser

6. Future Work

DEEP is planned to include tools that allow the user to see what is happening in the system as the parallel program executes (parallel program monitoring tools). The user will be able to visualize the current status of processors, message traffic, and calculations. These tools are currently in development.

Since DEEP supports both Data Parallel and SMP models, extending it to support hybrid/cluster/NUMA and other such systems seems to be a natural progression.

7. Acknowledgments

This work received support from DARPA.

DEEP is a trademark of Pacific-Sierra Research Corporation.

References

[1] *MPI-2: Extensions to the Message-Passing Interface*, Message Passing Interface Forum, July 18, 1997.
[2] *High Performance Fortran Language Specification*, HPF Forum, Version 2.0, 10/19/96
[3] *Data Parallel C Extensions*, Numerical C Extensions Group of X3J11, DPCE subcommittee, Version 1.6, X3J11/94-080, 12/31/94
[4] *VAST-DPC User's Guide*, Version 1.2, March 1997, Pacific-Sierra Research Corporation.
[5] *VAST-2/Parallel User's Guide*, Version 2.0, February 1998, Pacific-Sierra Research Corporation.
[6] *OpenMP Fortran Application Program Interface*, Version 1.0, October 1997

Venkatramani Balaji
Silicon Graphics/Cray Research
Geophysical Fluid Dynamics Laboratory
P.O. Box 308
Princeton NJ 08542
Phone Number: (609) 452-6516
Fax Number: (609) 987-5063
Email: vb@gfdl.gov

Tom Bettge
National Center for Atmospheric Research
1850 Table Mesa Drive
Boulder, CO 80303
Phone Number: (303) 497-1371
Fax Number: (303) 497-1348
Email: bettge@ucar.edu

Ilene Carpenter
Silicon Graphics/Cray Research
655E Lone Oak Drive
Eagan, MN 55121
Phone Number: (612) 683-3629
Fax Number: (612) 683-3699
Email: ilene@cray.com

Lang-Ping Chang
DAO, GSFC, NASA
Code 910.3
Goddard Space Flight Center
Greenbelt, MD 20771
Phone Number: (301) 805-6998
Fax Number: (301) 805-7960
Email: lpchang@dao.gsfc.nasa.gov

Samson Cheung
NASA/Ames Research Center
Mail Stop 258-6
Moffett Field, CA 94035
Phone Number: (650) 604-2875
Fax Number: (650) 604-4377
Email: cheung@nas.nasa.gov

Giri Chukkapalli
San Diego Supercomputer Center
P.O. Box 85608
San Diego, CA 92186-5608
Phone Number: (619) 534-5072
Fax Number: (619) 534-5117
Email: giri@sdsc.edu

Bertrand Denis
Canadian Centre for Climate Modelling and Analysis
P.O. Box 1700
Victoria, BC, Canada V8W 2Y2
Phone Number: (250) 363-8245
Fax Number: (250) 363-8247
Email: bdenis@ec.gc.ca

Mike Desgagne
RPN-AES, Environment Canada
2121 Trans-Canada Highway, 5th Floor
Dorval, Quebec, Canada H9P 1J3
Phone Number: (514) 421-4750
Fax Number: (514) 421-2106
Email: michel.desgagne@ec.gc.ca

Chris H.Q. Ding
National Energy Research Scientific Computing Center
Lawrence Berkeley National Laboratory, Bldg. 50F
Berkeley, CA 94720
Phone Number: (510) 486-6901
Fax Number: (510) 486-5548
Email: cding@nersc.gov

Jeff Durachta
International Business Machines
3624 Delverne Road
Baltimore, MD 21218
Phone Number: (410) 889-9967
Email: jdurachta@toad.net

Name: Gerald Embery
Bureau of Meteorology Australia
GPO Box 1289K
Melbourne, Victoria 3001, Australia
Phone Number: +61 3 9669-4417
Fax Number: +61 3 9669-4660
Email: G.Embery@bom.gov.au

John D. Farrara
University of California, Los Angeles
Department of Atmospheric Sciences
405 Hilgard Avenue
Los Angeles, CA 90095-1565
Phone Number: (310) 825-9205
Fax Number: (310) 206-5219
Email: jfarrara@ucla.edu

Jing Guo
NASA/Goddard Space Flight Center
Code 910.3
Greenbelt, MD 20771
Phone Number: (301) 805-8333
Fax Number: (301) 805-7960
Email: guo@dao.gsfc.nasa.gov

Steve Hammond
National Center for Atmospheric Research
P.O.Box 3000
Boulder, CO 80307
Phone Number: (303) 497-1811
Fax Number: (303) 497-1286
Email: hammond@ncar.ucar.edu

George Heburn
Naval Research Laboratory
NRL Code 7306
Washington, DC 20375
Phone Number: (202) 404-1437
Fax Number: (202) 404-1662
Email: heburn@nrl.navy.mil

Richard Hemler
Geophysical Fluid Dynamics Laboratory
P.O. Box 308
Princeton, NJ 08542
Phone Number: (609) 452-6598
Fax Number: (609) 987-5063
Email: rsh@gfdl.gov

Barry Herchenroder
SAIC/Goddard Space Flight Center
Ocean/Ice Branch, Bldg. 22
Greenbelt, MD 20771
Phone Number: (301) 286-5990
Fax Number: (301) 286-0240
Email: beh@janus.gsfc.nasa.gov

Mohamed Iskandarani
Institute of Marine and Coastal Sciences
Rutgers University
71 Dudley Road
New Brunwick, NJ 08901-8521
Phone Number: (732) 932-6555 x 251
Fax Number: (732) 932-8578
Email: mohamed@ahab.rutgers.edu

Rodney James
National Center for Atmospheric Research
P.O. Box 3000
Boulder, CO 80307
Phone Number: (303) 497-1271
Fax Number: (303) 497-1286
Email: rodney@ncar.ucar.edu

Jay Jayakumar
NAVO/MSRC PET Program
Bldg. 1103, Room 248
Stennis Space Center, MS 39529
Phone Number: (601) 688-3518
Fax Number: (601) 688-2764
Email: jkumar@navo.hpc.mil

Philip Jones
Los Alamos National Laboratory
T-3 MS B216
Los Alamos, NM 87545
Phone Number: (505) 667-6387
Fax Number: (505) 665-5926
Email: pwjones@lanl.gov

Wesley Jones
Silicon Graphics/Cray Research
2011 N. Shoreline, Bldg. MS 580
Mountain View, CA 94043
Phone Number: (650) 933-2992
FAX Number: (650) 932-2992
Email: wesley@sgi.com

Christopher L. Kerr
International Business Machines
Geophysical Fluid Dynamics Laboratory
P.O. Box 308
Princeton, NJ 08542
Phone Number: (609) 452-6573
Email: ck@gfdl.gov

Suraj C. Kothari
Iowa State University
Computer Science Department
Iowa State University, 207 Atanasoff Hall
Ames, IA 50011
Phone Number: (515) 294-7212
Fax Number: (515) 294-0258
Email: kothari@cs.iastate.edu

Jay Larson
University of Maryland
NASA/Goddard Space Flight Center
7501 Forbes Blvd., Suite 200
Seabrook, MD 20706
Phone Number: (301) 805-8334
Fax Number: (301) 805-7960
Email: jlarson@dao.gsfc.nasa.gov

John M. Levesque
Applied Parallel Research
1723 Professional Drive
Sacramento, CA 95825
Phone Number: (916) 481-9891
Fax Number: (916) 481-7924
Email: levesque@apri.com

Greg Lindahl
University of Virginia
Computer Science Department, Thornton Hall
Charlottesville, VA 22903
Phone Number: (804) 982-2293
Fax Number: (804) 982-2214
Email: lindahl@cs.virginia.edu

Luke Lonergan
High Performance Technologies, Inc.
1875 Campus Commons Drive, Suite 200
Reston, VA 20191-1533
Phone Number: (703) 262-0654
Fax Number: (703) 758-7866
Email: llonergan@hpti.com

Rob Lucchesi
Data Assimilation Office
NASA/Goddard Space Flight Center
Code 910.3
Greenbelt, MD 20771
Phone Number: (301) 286-9084
Fax Number: (301) 286-1754
Email: lucchesi@dao.gsfc.nasa.gov

Hong Ma
Brookhaven National Laboratory
Department of Applied Science, Bldg. 490D
Upton, NY 11973
Phone Number: (516) 344-4138
Fax Number: (516) 344-3911
Email: hm@bnl.gov

Bob Malone
Los Alamos National Laboratory
Advanced Computing Laboratory, MS B287
Los Alamos, NM 87545
Phone Number: (505) 667-5925
Fax Number: (505) 667-5921
Email: rcm@lanl.gov

Joe McCaffrey
Naval Research Laboratory
Code 7300
Stennis Space Center, MS 39529
Phone Number: (228) 688-5008
Fax Number: (228) 688-2764
Email: mccaffrey@nrlssc.navy.mil

James McGraw
Lawrence Livermore National Laboratory
P.O. Box 808, Mail Stop L-561
Livermore, CA 94550
Phone Number: (510) 422-0541
Fax Number: (510) 423-2993
Email: mcgraw1@llnl.gov

Anthony Meys
Silicon Graphics/Cray Research
655E Lone Oak Drive
Eagan, MN 55121
Phone Number: (612) 683-3426
Fax Number: (612) 683-3699
Email: meys@cray.com

John Michalakes
Argonne National Laboratory
Mathematics and Computer Science Division
9700 South Cass Avenue
Argonne, IL 60439
Phone Humber: (708)252-6646
Email: michalak@mcs.anl.gov

Clark Mobarry
NASA/Goddard Space Flight Center
Code 931
Greenbelt, MD 20771
Phone Number: (301) 286-2081
Fax Number: (301) 286-1634
Email: clark.mobarry@gsfc.nasa.gov

George Mozdzynski
European Center for Medium Range Forecasts
Shinfield Park
Reading, Berkshire RG2 9AX United Kingdom
Phone Number: +44-118-949-9113
Fax Number: +44-118-986-9450
Email: George.Mozdzynski@ecmwf.int

Philip Mucci
University of Tennessee
Department of Computer Science
Knoxville, TN 37996-1301
Phone Number: (423) 522-5211
Fax Number: (423) 974-8296
Email: mucci@cs.utk.edu

Robert Numrich
Silicon Graphics/Cray Research
655 E Lone Oak Drive
Eagan, MN 55121
Phone Number: (612) 683-5481
Fax Number: (612) 683-3699
Email: rwn@cray.com

Matthew O'Keefe
University of Minnesota
Department of Electrical and Computer Engineering
200 Union Street, S.E.
Minneapolis, MN 55455
Phone Number: (612) 625-6306
Fax Number: (612) 625-4583
Email: okeefe@ece.umn.edu

Mike O'Neill
Fujitsu Systems Europe
2 Longwalk Road, Stockley Park
Uxbridge, Middlesex
England, UB11 1AB
Phone Number: +44-181-606-4573
Fax Number: +44-181-606-4580
Email: meon@fujitsu.co.uk

Steve Piacsek
Naval Research Laboratory
Code 7320, NRL-SSC
Stennis Space Center, MS 39529
Phone Number: (601) 688-5316
Fax Number: (601) 688-4759
Email: piacsek@nrlssc.navy.mil

Kenneth Pollak
Fleet Numerical Meteorology and Oceanography Ctr.
7 Grace Hopper Avenue
Monterey, CA 93943
Phone Number: (408) 656-4335
Fax Number: (408) 656-4489
Email: kpollak@fnmoc.navy.mil

Joseph M. Prusa
Iowa State University
Department of Mechanical Engineering
Ames, IA 50011
Phone Number: (515) 294-0354
Fax Number: (515) 294-3261
Email: prusa@iastate.edu

Frederick Rawlins
UK Meteorological Office
Numerical Modelling Division
London Road
Bracknell, Berkshire RG12 2SZ United Kingdom
Phone Number: +44 01344 856482
Email: frawlins@meto.gov.uk

Tom Rosmond
Naval Research Laboratory
7 Grace Hopper Avenue, Stop 2
Monterey, CA 93943-5502
Phone Number: (408) 656-4736
Fax Number: (408) 656-4769
Email: rosmond@nrlmry.navy.mil

Bruce Ross
Geophysical Fluid Dynamics Laboratory
P.O. Box 308
Princeton, NJ 08542-0308
Phone Number: (609) 452-6504
Fax Number: (609) 987-5070
Email: br@gfdl.gov

Ulrich Schaettler
Deutscher Wetterdienst
Kaiserleistrasse 42
63067 Offenbach, Germany
Phone Number: +49-69-8062-2739
Fax Number: + 49-69-8236-1493
Email: uschaettler@dwd.d400.de

Daniel Schaffer
NASA/Goddard Space Flight Center
Code 971, Bldg. 22
Greenbelt, MD 20771
Phone Number: (301) 286-3133
Fax Number: (301) 286-0240
Email: Dans@janus.gsfc.nasa.gov

Jimmy Scott
Silicon Graphics, Canada
2550 Matheson Blvd. East
Mississauga, Ontario L4W 4Z1
Phone Number: (905) 282-8933
Fax Number: (905) 625-4476
Email: jcs@cray.com

Albert Semtner
Naval Postgraduate School
Oceanography Department
833 Dyer Road
Monterey, CA 93943
Phone Number: (408) 656-3267
Fax Number: (408) 656-2712
Email: sbert@ucar.edu

Piotr Smolarkiewicz
National Center for Atmospheric Research
P.O. Box 3000
Boulder, CO 80307
Phone Number: (303) 497-8972
Fax Number: (303) 497-8181
Email: smolar@ncar.ucar.edu

Gary Strand
National Center for Atmospheric Research
Climate and Global Dynamics Division/CCR
1850 Table Mesa Drive
Boulder CO 80303
Phone Number: (303) 497-1336
Fax Number: (303) 497-1348
Email: strandwg@ucar.edu

Max Suarez
NASA/Goddard Space Flight Center
Code 913
Greenbelt, MD 20771
Phone Number: (301) 286-7373
Fax Number: (301) 286-1759
Email: Max.Suarez@gsfc.nasa.gov

Dean Sumner
Los Alamos National Laboratory
MS D413
Los Alamos, NM 87545
Phone Number: (505) 667-0708
Fax Number: (505) 667-3726
Email: shd@lanl.gov

Steve Thomas
Recherche en Prevision Numerique
Environment Canada
2121 Trans-Canada Hwy, 5th Floor
Dorval, Quebec, Canada, H9P 1J3
Phone Number: (514) 421-4769
Fax Number: (514) 421-2106
Email: thomas@cerca.umontreal.ca

Alan Wallcraft
Naval Research Laboratory
NRL Code 7323
Stennis Space Center, MS 39529
Phone Number: (228) 688-4813
Fax Number: (228) 688-4759
Email: wallcraf@ajax.nrlssc.navy.mil

Chris R. Warber
Pacific-Sierra Research Corporation
2901 28th Street
Santa Monica, CA 90405
Phone Number: (310) 314-2340
Fax Number: (310) 314-2323
Email: cwarber@pacific-sierra.com

Vince Wayland
National Center for Atmospheric Research
P.O. Box 3000
Boulder, CO 80303
Phone Number: (303) 497-1300
Fax Number: (303) 497-1348
Email: wayland@ucar.edu

Patrick Worley
Oak Ridge National Laboratory
P.O. Box 2008, Bldg. 6012
Oak Ridge, TN 37831
Phone Number: (423) 574-3128
Fax Number: (423) 574-0680
Email: worleyph@ornl.gov

Andrzej Wyszogrodzki
National Center for Atmospheric Research
P.O. Box 3000
Boulder, CO 80307
Phone Number: (303) 497-8981
Fax Number: (303) 497-8181
Email: andii@ucar.edu

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	June 1998	Conference Publication

4. TITLE AND SUBTITLE
Second International Workshop on Software Engineering and Code Design in Parallel Meteorological and Oceanographic Applications

5. FUNDING NUMBERS
971

6. AUTHOR(S)
M. O'Keefe, C. Kerr, Editors

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS (ES)
Laboratory for Hydrospheric Processes
Oceans and Ice Branch
Goddard Space Flight Center
Greenbelt, Maryland 20771

8. PEFORMING ORGANIZATION REPORT NUMBER
98B00049

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS (ES)
National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
CP—1998–206860

11. SUPPLEMENTARY NOTES
O'Keefe: University of Minnesota; Kerr: International Business Machines. Workshop was cosponsored by the U.S. Dept. of Energy, Office of Biological and Environmental Research; U.S. Dept. of Defense, High Performance Computing and Modernization Office; NASA GSFC, Season-to-Interannual Prediction Project.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Unclassified–Unlimited
Subject Category: 61
Report available from the NASA Center for AeroSpace Information,
7121 Standard Drive, Hanover, MD 21076-1320. (301) 621-0390.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This report contains the abstracts and technical papers from the Second International Workshop on Software Engineering and Code Design in Parallel Meteorological and Oceanographic Applications, held June15–18, 1998, in Scottsdale, Arizona. The purpose of the workshop is to bring together software developers in meteorology and oceanography to discuss software engineering and code design issues for parallel architectures, including Massively Parallel Processors (MPP's), Parallel Vector Processors (PVP's), Symmetric Multi-Processors (SMP's), Distributed Shared Memory (DSM) multiprocessors, and clusters. Issues to be discussed include: (i) code architectures for current parallel models, including basic data structures, storage allocation, variable naming conventions, coding rules and styles, i/o and pre/post-processing of data; (ii) designing modular code; (iii) load balancing and domain decomposition; (iv) techniques that exploit parallelism efficiently yet hide the machine-related details from the programmer; (v) tools for making the programmer more productive; and (vi) the proliferation of programming models (F--, OpenMP, MPI, and HPF).

14. SUBJECT TERMS
Parallel architectures, Massively Parallel Processor, Parallel Vector Processor, Symmetric Multi-Processor, Distributed Shared Memory multiprocessor.

15. NUMBER OF PAGES
320

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL